A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms

Patrick David Surry

Doctor of Philosophy
University of Edinburgh
1998

To Sharon, for putting up with me and believing that it might one day be finished; and to Jean & David Surry, for a lifetime of encouragement to understand how things work.

Abstract

It has been widely recognised in the computational intelligence and machine learning communities that the key to understanding the behaviour of learning algorithms is to understand what *representation* is employed to capture and manipulate knowledge acquired during the learning process. However, traditional evolutionary algorithms have tended to employ a fixed representation space (binary strings), in order to allow the use of standardised genetic operators. This approach leads to complications for many problem domains, as it forces a somewhat artificial mapping between the problem variables and the canonical binary representation, especially when there are dependencies between problem variables (e.g. problems naturally defined over permutations). This often obscures the relationship between genetic structure and problem features, making it difficult to understand the actions of the standard genetic operators with reference to problem-specific structures. This thesis instead advocates making the representation of solutions the explicit focus, in order to highlight the way in which the genetic operators (and resulting search algorithms) form and test hypotheses about the relationship between observed problem structure and fitness.

It is clear that any search algorithm must limit the class of hypotheses which it is able to learn (its bias), if it is to select the most accurate of those hypotheses efficiently. We demonstrate this in the context of evolutionary search by exploring the so-called "no free lunch" results, and argue that it is the chosen representation which determines what kinds of hypotheses can be formed and tested by the algorithm. To do this, we exploit a general formalism for generating a representation for an arbitrary instance of a given problem domain, using a *characterisation* of that problem domain which captures beliefs about its structure. Such a characterisation is simply an explicit set of mathematical statements about the relationship between features of solutions and their fitness values, making it clear that the resulting representations encapsulate all of the domain knowledge which is available to any search algorithm.

We next present a method for specifying algorithms with respect to *abstract* representations, making them completely independent of any *actual* representation or problem domain. Such algorithms are based on generic genetic operators which can be used to manipulate solutions encoded using arbitrary representations. This allows us to specify mathematically precise yet completely problem-independent evolutionary algorithms. Such algorithms can be applied to any problem of interest by utilising any representation which has been generated for that problem to derive problem-specific forms of the

operators. This yields well-specified search strategies suitable for implementation. The process is demonstrated for several practical problem domains, showing for instance that *identical* algorithms can be applied to both the TSP and real parameter optimisation to yield familiar (but superficially very different) concrete search strategies.

Making representation central to the study of evolutionary algorithms allows us to interpret the actions of the genetic operators as manipulating hypotheses about problem structure. This motivates ideas about how to predict algorithmic performance and choose between alternative representations for a given problem domain, and begins to provide a clearer understanding of how evolutionary algorithms work.

Acknowledgements

The bulk of the material presented in this thesis has been directly adapted from a number of papers co-authored with Professor Nicholas Radcliffe (see below). It is largely he who has made this thesis possible, through the enthusiasm which spurred my own initial interest; his clarity of vision for the wider picture; his encouragement when no one else seemed to be listening; and above all the motivation, ideas, and words which have led to such a fulfilling collaboration.

My PhD studies have been primarily funded by Canada's National Science and Engineering Research Council through a "1967" award which allowed me the freedom to study outside Canada—at Edinburgh University in particular. I was also fortunate enough to obtain an Overseas Student Award from the British government during that time. First the Edinburgh Parallel Computing Centre and later Quadstone Limited have also been generous in their support: from the endless computing cycles eaten up by evolutionary algorithms through to the supportive environment in which to think and work.

Chapter 3 is structurally based on Radcliffe & Surry (1995), with an extended section on polynomial-time algorithms. The high-level approach of chapters 4–6 was first articulated in Surry & Radcliffe (1996a), although the presentation here is significantly expanded. The underlying components were developed earlier (for instance in Radcliffe & Surry, 1994b, and Radcliffe's earlier work). The practical illustrations of the techniques, in chapters 7, 8 and 9, are based on work on the travelling sales-rep problem (Radcliffe & Surry, 1994b), representations for real-parameter optimisation (Surry & Radcliffe, 1996c), and memetic algorithms (Radcliffe & Surry, 1994c) respectively. Again, additional material has been prepared for this thesis.

The work on inoculation and initialisation in chapter 10 is directly based on Surry & Radcliffe (1996b), and the experimental results reported there on credit scoring and oil-field production scheduling were made possible thanks to the assistance and encouragement of Alasdair Bruce and Dr. Timothy Harding respectively. The COMOGA approach of chapter 11 was first developed through study of the pipe-sizing problem reported in Surry *et al.* (1995) and later expanded in Surry & Radcliffe (1997).

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Patrick David Surry*)

Table of Contents

Glossary of Symbols

English Symbols

a, b, c	Generic elements of the search space
A, B, C	Generic sets
$\mathcal{A}$	A set of alleles
$\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$	Search algorithms
$B(n, p)$	The binomial distribution
$\mathcal{B}$	A neighbourhood set
$\mathbb{B}$	The binary set $\triangleq\{0, 1\}$
BLX-α	Blend crossover with parameter α
BMM	Binomial minimal mutation
$\mathcal{C}$	A set of chromosomes
$D(x, y)$	A distance metric
$\mathcal{D}$	A problem domain (a collection of instances)
E	A basis set of equivalence relations
$\mathbb{E}$	The universal set of equivalence relations
$\mathcal{E}$	A set of edges
$f(\cdot)$	A fitness function
$g(\cdot)$	A growth function
$\mathcal{G}$	A set of growth functions
GNX	Generalised N-point crossover
$\mathcal{H}$	A hill-climbing operator
I	A problem instance
k	A random seed
κ	A control parameter
$\mathcal{K}$	A set of control parameters
ℓ	A crossover point
ℓ_n	The effective length (of a chromosome)
$\mathcal{L}$	A set of crossover points
m	A memetic representation
M	A set of minimal mutations
$\mathcal{M}$	A mutation operator
$\hat{\mathcal{M}}$	A minimal mutation operator
n	The number of alleles in a chromosome
$o(\xi)$	The order of a forma ξ
p_X	The probability of crossover
p_I	The probability of inversion
p_M	The probability of mutation
$\mathcal{P}$	A set of Pareto-optimal solutions
$\mathbb{P}(A)$	The power set of the set A
$P\{\cdot\}$	The probability of an event

English Symbols (continued)

$\mathbb{Q}$	The rational numbers
$\mathcal{R}$	The range of an objective function
$\mathfrak{R}$	A reproductive function
$\mathbb{R}$	The real numbers
RAR	Random, assorting recombination
R^3	Random, respectful recombination
RTR	Random, transmitting recombination
$\mathcal{S}$	A search space
$\mathbb{S}(A)$	The (ordered) sequences over a set A
SEX	Strategic edge crossover
UX	Uniform crossover
x, y, z	Chromosomes in $\mathcal{C}$
X, Y, Z	Allelic chromosomes
$\mathcal{X}$	A crossover operator
$\mathbb{Z}$	The integers

Greek Symbols

α	An allele
γ	A member of the dynastic potential
$\Gamma(\cdot)$	The dynastic span of a set of solutions
ε	A basic equivalence relation
ζ	A basic forma
$\lambda(\cdot)$	A prospect function
$\mu(\cdot)$	A performance measure
ξ	A forma
Ξ	A set of formae
π	A permutation
Π	A set of permutations
$\rho(\cdot)$	A representation function
ϕ	An equivalence relation
χ	A characterisation of a problem class
ψ	An equivalence relation
Ψ	A set of equivalence relations
ω	A genetic move operator
Ω	A set of genetic move operators

Miscellaneous Symbols

$\equiv$	Is identically equal to
$\triangleq$	Is defined to be equal to
$\otimes$	Logical and
$\oplus$	Logical exclusive or
$\varnothing$	The empty set
$\square$	The "don't care" symbol
$\blacksquare$	The "care" symbol
$\langle\psi\rangle$	The description set of a forma ξ
$\lvert A\rvert$	The cardinality of a set A

List of Figures

List of Tables

Chapter 1

Introduction

To me our knowledge of the way things work, in society or in nature, comes trailing clouds of vagueness. Vast ills have followed a belief in certainty, whether historical inevitability, grand diplomatic designs, or extreme views on economic policy. When developing policy with wide effects for an individual or society, caution is needed because we cannot predict the consequences.

(Kenneth Arrow)

Many real-world applications demand the solution of highly complex search or optimisation problems, in which we seek the best element(s) from a set of candidate solutions according to a given performance metric. Often, the actual problem of interest is too difficult to solve directly or even to formulate precisely. In such cases a simplified, analytical model of the problem is typically constructed (perhaps only applicable within a particular parameter regime), that permits well-understood mathematical techniques such as linear or non-linear programming to be brought to bear. By limiting the complexity of the mathematical model, we are able to reduce the effort required to find optimal solutions, in the hope that results obtained by solving the simplified proxy can lead to useful generalisations or insights relevant to the actual problem of interest.

An alternative approach, of course, is to attack more accurate (and thus necessarily more complex) models of the problem directly. In this case, we must typically forego guarantees on quality of the resulting solution, and on the time required to find it, settling instead for probabilistic or heuristic bounds (Zhigljavsky, 1991).

Although problem-specific heuristics have been developed for many particular problems (e.g. Pearl, 1984), it is clearly attractive to consider the potential offered by generic, problem-independent algorithms. If effective problem-independent algorithms can be defined, they can be applied to each new problem domain that arises with the benefit of learnings gleaned from prior applications in other domains. In particular, the past twenty years has seen a rapid growth of interest in stochastic search and optimisation algorithms, particularly those inspired by natural processes in physics (Kirkpatrick *et al.*, 1983), cognition (Glover, 1995) and evolution (discussed at length below). These

have been found to be simple to implement, widely applicable to different problem domains, and robust to varying degrees in different areas (Bäck *et al.*, 1997).

1.1 Evolutionary algorithms for optimisation and search

We will concentrate here on so-called "evolutionary algorithms". Modelled on Darwin's insights into natural selection, such algorithms maintain a population of candidate solutions that are randomly mutated and recombined subject to selective pressures determined by some measure of their relative quality. Because the requirements for their application are so limited when compared to other techniques, they allow us to tackle almost arbitrarily complex search and optimisation tasks: for example, objective functions featuring noise, time-dependence, high-dimensionality, multi-modality, non-linearity or non-differentiability; essentially arbitrary constraints; multiple non-commensurate success criteria; and so forth. Although impressive results have been demonstrated on complex practical optimisation problems and related search applications taken from a variety of fields, the theoretical understanding of these algorithms remains weak.

As explored in chapter 2, most research focuses either on specific implementations for particular problems or on completely problem-independent theory. Studies of specific problems, although sometimes making dubious claims of sweeping generality, seem to offer little hope for transferable knowledge to other areas. This seems particularly clear when the wide range of suggested algorithmic variations, domain-specific move operators, and parameter choices is considered. On the other hand, the practical utility of theory not parameterised by any problem-dependent features (such as Markov chain analysis) is also far from obvious.

This results partly from the insufficient attention that has been paid to results showing certain fundamental limitations on universal search algorithms, including the so-called "No Free Lunch" Theorem (Wolpert & Macready, 1995). Chapter 3 extends these results and draws out some of their implications for the design of search algorithms, and for the construction of useful representations. The resulting insights focus attention on tailoring algorithms and representations to particular problem classes by exploiting domain knowledge. This highlights the fundamental importance of gaining a better theoretical grasp of the ways in which such knowledge may be systematically exploited when designing evolutionary algorithms. To quote from a recent survey of the state of evolutionary computation:

> *A constructive approach for the synthesis of evolutionary algorithms, i.e. the choice of design of the representations, variation operators, and selection mechanisms is needed.* (Bäck *et al.*, 1997)

A primary aim of this thesis is to ease the process of applying evolutionary algorithms to novel problem domains. Considering both that problems of interest are

typically "hard" (often formally NP-hard; Garey & Johnson, 1979), and the implications of the "No Free Lunch" theorem mentioned above, we argue that no single algorithm can be a panacea. Instead we focus on how externally available domain-knowledge is incorporated in the design of evolutionary algorithms. We demonstrate how to start from precise statements of beliefs about the relationship of problem structure to fitness and then mathematically derive a representation along with appropriate operators in order to construct a problem-specific evolutionary algorithm that exactly encapsulates those beliefs in its inductive bias. The hope, then, is to make the application of evolutionary algorithms to new problem domains rather more formulaic than the current "black art" (consider the spectrum of applications covered in the collections Grefenstette, 1985, 1987b; Schaffer, 1989; Belew & Booker, 1991; Forrest, 1993; Eshelman, 1995).

1.2 Representation in Evolutionary Search

Evolutionary algorithms are based on "genetic operators" that manipulate "chromosomes". These chromosomes are *representatives* of the structures in the actual search domain, and as such each have a "fitness" determined by the quality measure associated with the problem. We will show that the representation can be viewed as an interface through which a completely problem-independent algorithm can conduct (necessarily) problem-dependent search. In fact, the representation embodies all of the domain knowledge that can be exploited by the algorithm. By focusing attention on characterising beliefs about problem structure within the representation, we give a much more central rôle to the process of tailoring the algorithm to the problem (through making explicit beliefs about the problem domain) without sacrificing the generality and transferability of the algorithm itself.

The lack of attention to representation may be in part because it is often effectively transparent in simple cases: often, a "natural" representation for the problem exists so that the operators are viewed as acting "directly" in the space of candidate solutions and *not* via an intermediate representation. However, by making representation explicit we are led to examine the consequences of our decisions more carefully, challenging our previously implicit assumptions. Not only does this sometimes generate new ideas for the simple cases, but it provokes new insights into how evolutionary algorithms should be applied to more complex optimisation problems.

When considering the problem of how to represent a search domain in order to define evolutionary operators, most work has taken one of two approaches. In the first approach, a fixed "canonical" representation is used, and some mapping from the search space to the representation space is constructed (with the inverse mapping telling us to which solution each chromosome corresponds). In the second approach, operators

are designed specifically for each new search domain, and the problem of representation is largely ignored; conceptually the search operators work directly in the search space. Both of these approaches have significant drawbacks, and we propose an alternative methodology which captures the strengths of both while avoiding their weaknesses.

A canonical representation has the advantage that once operators (and hence algorithms) are defined, they can be applied to any new problem domain—all that is required is to define some mapping from the structures in the new search domain to, say, binary strings. Examples in which both genetic algorithms, which were first conceived for combinatorial problem domains, and evolution strategies, developed for continuous domains, have been coerced into other search domains are relatively commonplace (e.g. Wright, 1990; Groβ *et al.*, 1996). However, from the point of view of practical optimisation, this has an insurmountable drawback. Recent work (Wolpert & Macready, 1995; Radcliffe & Surry, 1995) has confirmed what has been known intuitively for some time—that the representation of the search domain must capture, in some way, the structure of the objective function in order for there to be any possibility of out-performing enumeration. Simply transferring algorithms between problems using a fixed representation space is doomed to failure *unless* the representational mapping is carefully constructed (or accidentally chances) to preserve search-space structure in a form that can be exploited by the genetic operators: no algorithm can be an effective black-box optimiser.

A problem-specific approach avoids these difficulties, as *ad hoc* operators can be defined to exploit known characteristics of the problem at hand, or "standard" operators can be modified by forcing them to make moves that appear "sensible" within the search space. However, this approach has the clear disadvantage that work is not easily transferable to new search domains.

1.3 A Constructive Approach to Designing Evolutionary Algorithms

We argue that a middle ground is preferable, recognising the need for problem-specific knowledge but preserving the definition of an algorithm across disparate problem domains. A methodology based on *forma analysis* (Radcliffe, 1994) is presented in chapter 4 that allows an appropriate representation for a given search domain to be generated directly from statements of belief about the search space. The theory is based on characterising beliefs about the structure of a domain of optimisation problems, typically by identifying features of solutions thought to be related to performance. This characterisation mathematically generates a representation space and growth function for any given instance of the problem. Individual solutions are represented by chromosomes that indicate a number of *formae* (generalised schemata, equivalent to subsets of the

search space) to which they belong.

Once the representation has been chosen, problem-specific forms of universal "representation-independent" genetic operators can be derived mathematically. This is important because many of the representations generated by explicitly characterising beliefs about the structure of the search problem turn out to be *non-orthogonal* (in which not all combinations of alleles are legal), meaning that "traditional" genetic operators cannot be used without resorting to corrective mechanisms such as repair, penalty functions, etc. (Consider, for example, trying to use N-point crossover on two permutations: invalid solutions typically result).

This framework enables the separation of algorithm and domain-knowledge to be made completely explicit. Representation-independent search algorithms (constructed with generalised move operators) can be precisely specified mathematically. For a given search domain, beliefs about its structure are mathematically formalised and a well-defined procedure is then employed to construct a representation. This representation is then used to instantiate the generalised algorithm to derive a computationally implementable, problem-specific search strategy.

Note in particular that the significance of the choice of representation is *not* primarily the way in which real values are physically stored in a digital computer (which is ultimately always as bit patterns), but rather, the way in which it affects the moves effected in the search space by the chosen genetic operators. Representation, as discussed here, is simply a mathematical device for deriving domain-specific operators that incorporate our explicit beliefs about problem structure.

Chapter 5 details how such *formal algorithms* can be precisely specified—completely independently of any particular representation or problem domain—in such a way that their effectiveness is a direct function of the quality of the domain knowledge captured by the allele structure generated by the characterisation. The main ingredient required here is a set of move operators that can be applied in arbitrary problem domains, since many problem domains are naturally characterised using non-orthogonal representations in which traditional operators are not applicable. In order to facilitate this, move operators are defined that manipulate solutions solely on the basis of their abstracted subset-membership properties, rather than acting on problem-specific features. We provide examples of generalisations of N-point and uniform crossover, and of mutation and hill-climbing operators, and later show how they reduce to traditional forms in familiar problem domains.

We proceed in chapter 6 to show how *any* (appropriate) formal algorithm can be instantiated with *any* suitable representation of a problem domain of interest to produce a concrete search algorithm that we will term a *search strategy*. This is illustrated by defining a simple representation-independent evolutionary algorithm and instantiating

it on the one hand to solve the TSP and on the other to solve a real-parameter optimisation problem. The resulting search strategies for the two problems look very different from each other but are both similar to evolutionary algorithms commonly applied in their respective domains. We thus prove the surprising result that two apparently quite different algorithms, in two completely different problem domains, are in fact *identical,* in a strong mathematical sense.

To summarise the implications of this more formal approach, we see that by separating algorithm and representation, we achieve the goal of truly problem-independent algorithms. This separation also makes the rôle of domain knowledge in the search process much more explicit, allowing us to pose more carefully questions such as "What is a good algorithm given certain properties of the characterisation?" and "What is a good characterisation of a given problem domain?" Although this formalism might be argued to contain a certain degree of circularity, it is seen to yield practical benefits. For instance, we are able to transfer such algorithms between arbitrary problem domains and to compare different algorithms fairly, over a range of disparate problem domains.

Once a formal representation-independent algorithm has been selected, the performance of alternative representations for a given problem domain can be measured. Simple analyses suggest that fitness variance of formae (generalised schemata) for the chosen representation might act as a performance predictor for evolutionary algorithms. This hypothesis is tested and supported in chapter 7 through studies of four different representations for the travelling sales-rep problem (TSP) in the context of formal representation-independent genetic algorithms (extended in chapter 9 to corresponding memetic algorithms).

Chapter 8 examines evolutionary optimisation of real-parameter functions, within the formal framework developed in chapters 4–6. Two new representations for real-parameter spaces are introduced—the Dedekind and Isodedekind representations. Point mutation and uniform crossover—in their generalised, representation-independent form—are shown, when instantiated with respect to these representations, to give rise to familiar operators for continuous domains, such as gaussian mutation, blend crossover and line recombination. Furthermore, the abstract concept of forma variance discussed in chapter 6 is shown to be closed related to a standard Lipschitz assumption in the real-parameter domain. Both the Dedekind and Isodedekind representations are highly non-orthogonal (admitting many illegal chromosomes), but, as is demonstrated, this causes no practical or theoretical problems. Moreover, these novel representations are shown to have sensible behaviour as the continuous limit is taken, while both "traditional" integer- and Gray-coding (Caruana & Schaffer, 1988) are shown to exhibit pathological behaviour.

the universal applicability of penalty-function approaches, but requires significantly fewer free control parameters.

COMOGA takes a dual perspective, considering a constrained optimisation problem sometimes as a constraint satisfaction problem, and sometimes as an unconstrained optimisation problem. These two formulations are treated simultaneously, using a single population, by basing each selection decision on the basis of either constraint violation or function value. A simple adaptive feedback mechanism couples the two formulations by adjusting the relative likelihood of these choices. Unlike penalty function approaches, COMOGA dynamically adapts the emphasis placed on constraint satisfaction and objective function value as the optimisation proceeds, yielding final populations which are both feasible and highly fit.

The method has been successfully applied to real industrial problems with comparable performance to highly tuned penalty function approaches. On a test suite of constrained problems previously studied by Michalewicz (1995a), application of COMOGA required minimal effort but proved superior to all previous evolutionary methods known to have been applied; indeed it was the only method which found feasible solutions in every run for every problem.

1.5 Discussion

The issues explored in this thesis are central to gaining greater understanding of (evolutionary) search methods, yet the questions that they raise receive remarkably little attention within the research community. It is hoped that the work reported here will in some small way begin to redress this balance, and will also stimulate further investigation. In chapter 12 we discuss what we consider to be key areas of focus for such research.

Chapter 2

Survey

Most gulls don't bother to learn more than the simplest facts of flight—how to get from shore to food and back again. (Richard Bach)

The individual sections of this thesis are relatively self-contained and consequently are presented with close reference to relevant existing work. As such, this chapter seeks only to provide a broad overview of the literature as it pertains to the main issues tackled in the thesis. For a more detailed introductory treatment, the reader is referred to the excellent survey of Bäck *et al.* (1997).

In broad terms, the theoretical basis of evolutionary computation remains relatively poorly developed. Early work in the field strayed more towards implementing algorithms than to developing a coherent theory underpinning them (a notable exception being the seminal work of Holland, 1975). This bias toward exploring practical applications of powerful ideas from evolutionary biology is understandable in a young and exciting new area. It seems unfortunate, however, that for the most part there has been a failure to build a suitable foundational theory to capitalise on this early (practical) work—even today significant effort is devoted to experimental study of "toy" problems with dubious relevance to practical areas of optimisation (Davis, 1991a).

There are several factors that can be identified as contributing to this state of affairs, including: the isolated way in which different branches of the field developed initially; an over-zealous acceptance of the first theoretical directions; the ease and attraction of exploring biological parallels empirically; and the complexity of formalising the interplay between highly stochastic algorithms and the difficult practical problems being tackled.

The development of evolutionary computation was characterised by strikingly compartmentalised and isolated early research. In fact, three separate strands developed largely independently for ten to fifteen years, only recently coalescing into what is today collectively termed evolutionary computation (Bäck & Schwefel, 1993). On the east coast of America, Holland (1975) pioneered the study of *genetic algorithms* (later made accessible to a wider audience by Goldberg, 1989c). This focused almost exclusively on evolving bit-string chromosomes to solve optimisation problems. (A recently

identifiable descendant of this work arising on the west coast is *genetic programming*, popularised by Koza, 1990, 1991, 1992, 1994; which considers the evolution of functions represented as operator-operand trees.) In Germany, a school of evolutionary algorithms grew out of work on *evolution strategies* by Rechenberg (1973, 1984) and Schwefel (1981). This was initially driven by optimisation of physical experiments on engineering-design problems, and has tended to concentrate on functions of real-valued parameters. The third strand, arising on the west coast of America, is due to Fogel *et al.* (1966) who considered the evolution of finite state machines as a mechanism to create artificial intelligence, now commonly known as *evolutionary programming*.

It is instructive to compare the relative state of theoretical development in the various branches of evolutionary computation. In genetic programming, for example, there is very little explanatory theory. This is perhaps understandable given the complexity of the representation, operators and problem-domain itself: genetic operators manipulate tree-based representations of arbitrary functions, with highly non-linear interactions between their actions and the resulting fitness values. Although analogues of the Schema Theorem of genetic algorithms (Holland, 1975) have been developed (see for example the survey in Poli & Langdon, 1998), the workings of these algorithms are poorly understood at best. Initial study in evolutionary programming shared a similar difficulty as the finite-state machines being evolved induced a highly complex relationship between genetic actions and fitness. (Today, however, evolutionary programming techniques are applied more widely in real-parameter optimisation, resulting in convergence with the theory of evolution strategies.)

In sharp contrast, there is a relatively large body of theoretical work supporting evolution strategies. Here there has always been a strong focus on the fixed problem domain of real-parameter optimisation, with results built on the large body of work on other optimisation techniques within this domain (e.g. Box *et al.*, 1969). This has encompassed theory on convergence properties for simple functions like quadratic forms; investigation of the relationship between time-complexity and function dimensionality; and control of endogenous parameters such as the mutation rates and rotation angles for the control variables (Bäck *et al.*, 1991). Related algorithms amenable to theoretical study have also been developed. For instance, the Breeder Genetic Algorithm of Mühlenbein & Schlierkamp-Voosen (1993) is based on an artificial breeding model, rather than the traditional natural selection schemes.

In genetic algorithms, the situation is somewhat clouded—although there is a significant body of theoretical work, there is little that supports useful, testable predictions about algorithmic behaviour in practical circumstances. Holland (1975) originally formulated genetic search as a decision-making problem of optimum allocation of trials to the levers of a k-armed bandit with the aim of maximising expected accumulated

profit (providing a simple model of the exploration versus exploitation dilemma). In this scheme, template bit patterns called schemata play the rôle of the arms of the bandit and observed fitness the rôle of profit. Apart from being a somewhat questionable model of optimisation (in which one is typically interested in the maximum observed fitness over time rather than the average), various workers (Grefenstette & Baker, 1989; Macready & Wolpert, 1996) have cast doubt on the use of k-armed bandits to justify the exponential allocation of trials claimed as the key to implicit (*née* intrinsic) parallelism.

Nevertheless, the resulting schema analysis (based around Holland's schema theorem) has dominated subsequent theoretical work on genetic algorithms. For instance, Bridges & Goldberg (1987) first wrote down a version of the schema theorem that replaces its inequality with an equality by considering creation of schema instances as well as their destruction. The resulting form has been used to build "executable" models of genetic algorithms. For example, Whitley (1992) simulated the exact behaviour (in expectation) of a simple genetic algorithm without mutation. Similarly, study of genetic algorithms with infinite populations (Liepins & Vose, 1990; Nix & Vose, 1991; Vose & Liepins, 1991a; Vose, 1992), is theoretically attractive because the algorithm becomes deterministic—with an infinite population the expected next population at each step is always achieved exactly. Furthermore, finite population genetic algorithms can be thought of as stochastically generating a discrete approximation of the population that is produced by the infinite population model at each step. Vose describes a genetic algorithm (with finite or infinite population size) using the full transition matrix between all possible population states, $\mathcal{G}$, that maps the current population to the expected next population. Analytical study of the properties of $\mathcal{G}$ is then used to determine the algorithm's expected population dynamics. Although it is clear that for sufficiently large population sizes an algorithm's behaviour can be approximated closely by that derived analytically for the infinite case, the relationship between population size and approximation accuracy is unknown. (That is, if we want to choose the population size N for which an infinite population model will predict the behaviour of the finite-population algorithm, such that the population distributions agree within a tolerance ϵ for t timesteps with probability p, what is the function $N(\epsilon, t, p)$?) Because these models require that the entire transition matrix between population states be computed, they are only presently capable of being used to analyse very small search spaces (typically binary problems with no more than a few bits). Various Markov models of evolutionary algorithm behaviour have also been developed (e.g. Eiben *et al.*, 1990; Davis & Principe, 1993), but it is not clear how relevant these are to practical optimisation (limited to finite populations and finite time).

A large body of work has also been generated from the study of "deception", which essentially aims to understand what makes particular combinations of objective function

and representation difficult for simple genetic algorithms to search efficiently. This is seen by some (e.g. Whitley, 1992) as the central problem in genetic search, and by others as more peripheral (e.g. Grefenstette, 1992). The term itself, first defined by Goldberg (1989a, 1989b), is typically used loosely to mean that the static *building block* hypothesis is violated, i.e. that the static average fitnesses of competing low-order schemata "point the wrong way" when used to infer which higher-order schemata contain better solutions. The work is based on Walsh analysis, introduced by Bethke (1980), that allows the fitness of a solution to be written be decomposed as a sum of contributions for each of the (binary) schemata to which it belongs (later extended by Mason (1991) to more general partition functions). It is claimed, by appealing to the schema theorem, that those schemata with the highest (disruption-adjusted) fitnesses are those which would attract the most trials as a genetic search proceeded. Goldberg acknowledges that this static analysis "ignores all dynamical and stochastic considerations", but asserts that it is a reasonable starting point for analysis. The claim allows deception to be interpreted as meaning that genetic search would be led away from some of the near-optimal points in the search space. However, Grefenstette (1992) has shown that aspirations that such static analyses are relevant to the dynamic characteristics of genetic search may be unreasonable, since these static averages can be arbitrarily bad estimators of the dynamically *observed* averages that actually govern the course of the search.

As noted above, because such theory is not directly related to measurable properties of the optimisation problems themselves, there are serious question marks over its practical utility. This should be compared to the theory of related stochastic optimisation techniques such as simulated annealing (van Laarhoven & Aarts, 1989) in which convergence results based on measurable problem characteristics have been established, or to the results for real-parameter optimisation on global random search (Zhigljavsky, 1991), recently shown to encompass certain classes of genetic algorithms (Peck & Dhawan, 1995).

Attempts to employ genetic algorithms in application areas where the "universal" binary chromosome is less than ideal have begun to focus attention on representation as a key issue. For instance, use of genetic algorithms to tackle optimisation of real-parameter functions has led to investigations of different binary representations (the so-called Gray coding debate; Caruana & Schaffer, 1988; though see Culberson, 1996, who shows that this argument is in fact futile without additional information), and discussion about whether a real-valued coding might be more natural in this domain (with, e.g. Davis, 1991b, arguing for, and Goldberg, 1990, against). Applications like the travelling salesman problem, with naturally non-orthogonal representations, have also highlighted this issue (e.g. Grefenstette *et al.*, 1985). This led to practical invest-

igations of various aspects of representation. Shaefer (1987) developed the adaptive representation genetic optimizer technique (ARGOT) which aimed to combine traditional Darwinian evolution with Lamarckian learning of "good" representations. It was based on dynamic adaptation of the representation based on measures of gene-wise convergence and population diversity to avoid premature convergence in real optimisation, and was extended in Shaefer & Smith (1990) to combinatorial problems. Other studies of dynamically changing representation include the work of Whitley *et al.* (1991a) on delta coding, in which the resolution of the representation is increased in "interesting" regions of the search domain; the conceptually similar dynamic parameter encoding approach of Schraudolph & Belew (1990), and the study of dynamic base changes in Kingdon & Dekker (1995). More generally Schmidhuber (1995) investigates learning strategies which are themselves evolved to learn more effectively.

In an attempt to emphasise the rôle of representation in search, Radcliffe (1991a, 1994) has developed *forma analysis* which extends schema analysis to domains in which traditional orthogonal genetic representations are inapplicable or inappropriate. This underpins the formal approach of this thesis, in which we seek to link explicitly the performance of the optimisation algorithm to measurable characteristics of the problem domain. Recent efforts in a similar vein include the statistical mechanical treatment of Shapiro *et al.* (1994), Prügel-Bennett & Shapiro (1995) and Rattray (1995). They have developed a macroscopic approach for modelling genetic algorithms based on the evolution of population fitness and correlation cumulants. While this does not have the precision of Vose's detailed models, it has the advantage of computational tractability and may lead to the development of accurate models of algorithmic behaviour for realistic problems, potentially by exploiting the notion of forma variance presented in chapter 6. Other authors have proposed similar, though perhaps more pragmatic approaches: Grefenstette (1995) describes a virtual genetic algorithm that attempts to model performance based on *post facto* measurements of trial algorithm runs; and Jones & Forrest (1995) show that *fitness-distance correlation*, which measures how closely the fitness of solutions is correlated with their operator-induced distance to the closest global optimum, is a good indicator of genetic algorithm performance.

The failure to recognise the importance of representation earlier seems odd, particularly in light of the strong emphasis on representation in related fields of computer science such as machine learning, and early cross-disciplinary work (e.g. McKinnon, 1972a). These fields have established a strong theoretical basis by recognising that algorithmic properties can only be established with reference to problem characteristics:

> *The richer the representation, the more useful it is for subsequent problem-solving, [but] the more difficult it is to learn. [...] This observation has led computational intelligence researchers to make representations the primary focus of study.* (Poole *et al.*, 1998)

Although there has been much recent interest in the so-called "No Free Lunch" theorem (Wolpert & Macready, 1995; Radcliffe & Surry, 1995) within the evolutionary computation community, analogous results were established far earlier in the artificial intelligence community (Watanabe, 1969). As English (1997) points out while providing an information-theoretic demonstration of No Free Lunch, it is intuitively clear that no algorithm can gain information about the fitness of unvisited points *unless* an inference can be made that combines information about visited points with meta-knowledge about the likely problem structure.

> *The ability to make an appropriate "inductive leap" when generalizing from a small set of training instances is possible only under a priori biases for choosing an appropriate generalisation out of the many possible.*
>
> (Mitchell, 1980)

In a machine learning environment, Mitchell (1980) suggests that such knowledge might include factual knowledge of the domain; bias towards simplicity and generality; intended use of the learned generalisations; knowledge about the source of the training data (e.g. an organised curriculum in supervised learning); and analogy with previously learned generalisations. Optimisation algorithms can clearly benefit from at least the first two.

In summary, there appears to be scope for a productive synthesis between this kind of theory and that of evolutionary computation. For instance, Jones (1995) links evolutionary optimisation with the concept of heuristic state-space search in artificial intelligence, by considering the fitness landscape as a graph with transition probabilities induced by genetic operators. Culberson (1996) explores the No Free Lunch result in relation to computational complexity theory, and speculates about the the possibility of using such techniques in the study of evolutionary algorithms. However, there is clearly more to be done. It is hoped that this thesis goes some way towards establishing a more productive context for the development of such a practical theory for evolutionary algorithm design and performance prediction.

Chapter 3

Stochastic Search and Optimisation

You should not have a favourite weapon. To become over-familiar with one weapon is as much a fault as not knowing it sufficiently well.

(Miyamoto Musashi)

The last two decades has seen increasing interest in the application of stochastic search techniques to difficult problem domains. Strong biological metaphors led to the development of several schools of evolutionary computation, including work on genetic algorithms and classifier systems (Holland, 1975), evolution strategies (Rechenberg, 1973, 1984; Bäck & Schwefel, 1993), evolutionary programming (Fogel *et al.*, 1966) and genetic programming (Koza, 1991). Physical analogues provided both the motivation and theoretical framework for the method of simulated annealing (Kirkpatrick *et al.*, 1983). Other randomised methods, such as tabu search (Glover, 1986) and a variety of stochastic hill climbers and related search techniques have also attracted much interest.

Although some theoretical progress has been made in the study of these and other stochastic search techniques, most attention has focused on the artificial situation of arbitrary ("black-box") search problems. Despite occasional warnings from various researchers (Vose & Liepins, 1991b; Radcliffe, 1992b, 1994; Wolpert & Macready, 1995) a great deal of research seems oblivious to the fact that in such a situation there is no scope for distinguishing any of these biased sampling methods from enumeration—random or fixed—as we demonstrate below. There are exceptions to this, for example the convergence results for simulated annealing which exploit knowledge of the problem domain (e.g. van Laarhoven & Aarts, 1989, Shapiro *et al.*, 1994) but much effort continues to be devoted to the "general" case, (Vose, 1992; Goldberg, 1989c, 1989a, 1989b). While such studies lead to elegant mathematics, and provide occasional insights into stochastic search, their practical utility in guiding—or even making contact with—practitioners tackling real search problems remains to be demonstrated.

This chapter explicitly demonstrates these fundamental limitations on search algorithms, and highlights the necessity of theory which incorporates knowledge about

the problem domain of interest. Much interest in this topic has been aroused lately as awareness has grown of the so-called "No Free Lunch Theorem" (Wolpert & Macready, 1995; 1997), which—broadly—states that if a sufficiently inclusive class of problems is considered, no search method can outperform an enumeration. We address the consequences of this and other fundamental limitations on search for the practical design of stochastic search algorithms, particularly evolutionary algorithms, and emphasize that there are many possible improvements that are *not* strongly limited by the theorems.

The discussion is first set in context by reviewing and formalising key concepts relating to search, concentrating on representation, search algorithms and performance measures. We then proceed to present a more accessible and general form of the No Free Lunch Theorem, based on arguments put forward previously (Radcliffe, 1992b, 1994). Strictly, this result shows that over the ensemble of all representations of one space with another, all algorithms (in a rather broad class) perform identically on any reasonable measure of performance. This has as an immediate corollary the No Free Lunch Theorem, and provides an interesting context for the "minimax" results given in Wolpert & Macready (1995). This result is then extended to practical (polynomial-time) algorithms.

Having proved the main results, attention turns to a discussion of their implications. First, issues concerning searches that re-sample points previously visited are discussed. After this, representational issues for tackling restricted problem classes are considered. This focuses on how knowledge concerning the structure of some class of functions under consideration can be used to choose an effective algorithm (consisting of a representation, a set of move operators and a sampling strategy). In particular, the critical rôle of representation in determining the potential efficacy of search algorithms is once again highlighted.

3.1 Search Concepts

Before formulating theorems about search, it will be useful to introduce some terminology, to reduce the scope for misinterpretation. As illustrated in figure 3.1, a *search problem* is taken to be the task of attempting to solve any problem instance from a well-specified problem domain, where by solve we mean finding some optimal or near-optimal solution from a set of candidate solutions. A *problem domain*, $\mathcal{D}$, is considered here to consist of a set of *problem instances*, I, each of which takes the form of a *search space*, $\mathcal{S}$—the set of candidate solutions over which the search is to be conducted—together with an *objective function* f, which is a mapping from $\mathcal{S}$ to the space of *objective function values*, $\mathcal{R}$, i.e.

$$f : \mathcal{S} \longrightarrow \mathcal{R}. \tag{3.1}$$

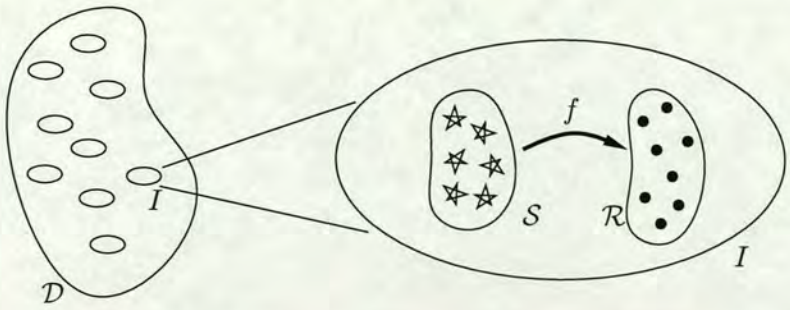


Figure 3.1: A *problem domain* $\mathcal{D}$ consists of a set of *problem instances*. Each instance I defines a *search space* $\mathcal{S}$ of candidate solutions, a *fitness function* f and a set of objective values $\mathcal{R}$.

In the bulk of this thesis, we will assume that $\mathcal{S}$ is finite, though discuss the case of an infinite search space ($\mathbb{R}^n$) in chapter 8.

Although the most familiar situation arises when $\mathcal{R} = \mathbb{R}$ (the set of real numbers) and the goal is minimisation or maximisation of f, it is not necessary to require this for present purposes. For example, $\mathcal{R}$ could instead be a vector of objective function values, giving rise to a multi-criterion optimisation. Note that the set of all mappings from $\mathcal{S}$ to $\mathcal{R}$ will be denoted $\mathcal{R}^{\mathcal{S}}$, so $f \in \mathcal{R}^{\mathcal{S}}$.

For instance, "symmetric travelling sales-rep problems" is a problem domain, with a particular set of $n(n-1)/2$ inter-city distances defining an instance (yielding a search space of $(n-1)!/2$ tours and an associated fitness function).

A *search strategy* is then simply a prescription by which, for any problem instance, we successively sample candidate solutions from the search space, typically biasing the samples depending on the observed quality (fitness) of previously sampled points. Any such strategy can be viewed as utilising one or more *move operators* which produce new candidate solutions from those previously visited. This idea is explored in more detail in section 3.7.

3.2 Representation

Because the objects in the search space can be arbitrary structures (e.g. real-valued vectors, TSP tours, neural-network topologies, etc.), it is often helpful to define search algorithms with respect to an abstract *representation* of the search space, allowing the transfer of search algorithms between problem domains. In general, a representation consists of a *representation space* and a *growth function*. The representation space defines a set of *chromosomes* which will be manipulated (by the move operators) during search, and the growth function defines a mapping between chromosomes and solutions.

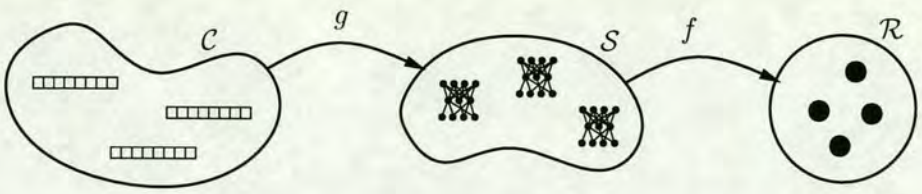


Figure 3.2: The *growth function* g maps the *representation space* $\mathcal{C}$ onto the *search space* $\mathcal{S}$, which is mapped to the space of *objective function values* $\mathcal{R}$ by the *objective function* f. If g is invertible, the representation is said to be *faithful.*

In this work, a representation of $\mathcal{S}$ will be taken to be an enumerable set $\mathcal{C}$ (the *representation space*, or the space of *chromosomes*) of size at least equal to $\mathcal{S}$, together with a *surjective* function g (the *growth function*) that maps $\mathcal{C}$ onto $\mathcal{S}$:

$$g : \mathcal{C} \longrightarrow \mathcal{S}. \tag{3.2}$$

(Surjectivity is simply the requirement that g maps at least one point in $\mathcal{C}$ to each point in $\mathcal{S}$.) The set of surjective functions from $\mathcal{C}$ to $\mathcal{S}$ will be denoted $\mathcal{S}^{\mathcal{C}}_{>}$, so $g \in \mathcal{S}^{\mathcal{C}}_{>} \subset \mathcal{S}^{\mathcal{C}}$. We will sometimes also consider the *representation function* ρ defined by

$$\rho : \mathcal{S} \longrightarrow \mathcal{C} \tag{3.3}$$

which is typically required to be injective (so that every solution $a \in \mathcal{S}$ has a well-defined (and unique) chromosome $\rho(a) \in \mathcal{C}$ that represents it). The representation function is related to the inverse of the growth function, but the two are not necessarily equivalent (if ρ is not surjective, there may be chromosomes in $\mathcal{C}$ which do not correspond to any solution in $\mathcal{S}$; and if g is not injective then g^{-1} is not a function at all, as there can be multiple chromosomes representing a single solution). If $\mathcal{C}$ and $\mathcal{S}$ have the same size, then $g \equiv \rho^{-1}$ and is clearly invertible, and will be said to be a *faithful representation.*

Objective function values can be associated with points in the representation space $\mathcal{C}$ through composition of f with g,

$$f \circ g : \mathcal{C} \longrightarrow \mathcal{R} \tag{3.4}$$

being given by

$$f \circ g(x) \triangleq f(g(x)). \tag{3.5}$$

The relationship between $\mathcal{C}$, $\mathcal{S}$ and $\mathcal{R}$ is shown in figure 3.2.

Although the present section has taken for granted that a representation of some sort will be used during search, it is clear that we could conceptually search 'directly' in the search space by taking $\mathcal{C} = \mathcal{S}$ with g the identity mapping. However, as arguments in this chapter will once again confirm, the issue of selecting an appropriate representation is central to search, and further motivation for why this should be so is given in section 3.7.

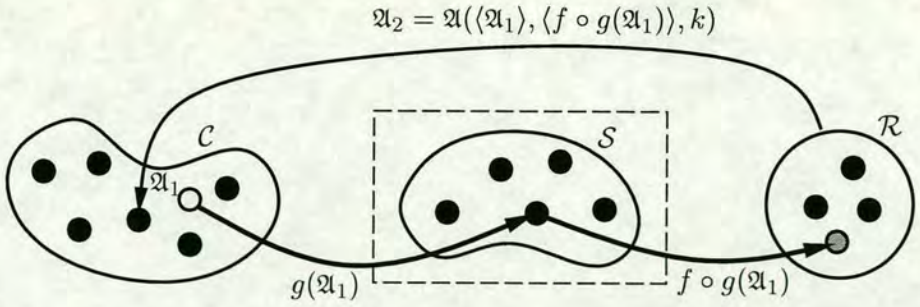


Figure 3.3: The first point in the search is determined by the algorithm and the seed, $\mathfrak{A}_1 = \mathfrak{A}(\langle\rangle, \langle\rangle, k)$. The next point, $\mathfrak{A}_2$ is determined by these values together with the new information $f \circ g(\mathfrak{A}_1)$. Note that the algorithm has no information about the point $g(\mathfrak{A}_1)$ in $\mathcal{S}$ which was used to generate the observed function value: this is entirely controlled by the representation g.

3.3 Search Algorithms

For present purposes, we will consider a somewhat technical definition of search algorithm—a *deterministic search algorithm* will be defined by a function which, given a sequence[1] of points $\langle x_i \rangle$, with each $x_i \in \mathcal{C}$, and the corresponding sequence of objective function values for those points under the composed objective function $f \circ g$, generates a new point $x_{n+1} \in \mathcal{C}$. Thus, a search algorithm is defined by a mapping

$$\mathfrak{A} : \mathbb{S}(\mathcal{C}) \times \mathbb{S}(\mathcal{R}) \longrightarrow \mathcal{C}. \tag{3.6}$$

The first point in the search sequence $\langle \mathfrak{A}_i \rangle \in \mathbb{S}(\mathcal{C})$ associated with the algorithm defined by $\mathfrak{A}$ is

$$\mathfrak{A}_1 \triangleq \mathfrak{A}(\langle\rangle, \langle\rangle), \tag{3.7}$$

with subsequent points being recursively defined by

$$\mathfrak{A}_{n+1} \triangleq \mathfrak{A}(\langle \mathfrak{A}_i \rangle_{i=1}^{n}, \langle f \circ g(\mathfrak{A}_i) \rangle_{i=1}^{n}). \tag{3.8}$$

Note that we shall tend to abuse the notation slightly by identifying a search algorithm with the function that defines it, i.e. the search algorithm defined by the mapping $\mathfrak{A}$ will also be denoted $\mathfrak{A}$.

A *stochastic search algorithm* will be taken to be a deterministic search algorithm augmented by a a seed for a pseudo-random number generator that is used to influence its choice of the next point x_{n+1}. Formally, a stochastic search algorithm is then defined by a mapping $\mathfrak{A}'$ of the form

$$\mathfrak{A}' : \mathbb{S}(\mathcal{C}) \times \mathbb{S}(\mathcal{R}) \times \mathbb{Z} \longrightarrow \mathcal{C}. \tag{3.9}$$

[1] Given a set A, an ordered sequence of its elements will be indicated by angle brackets, for example $\langle a_1, a_2, \ldots \rangle$. Further, $\mathbb{S}(A)$ will denote the set of all finite sequences over A, including the empty sequence $\langle\rangle$, i.e. $\mathbb{S}(A) \triangleq \left\{ \langle a_1, a_2, \ldots, a_n \rangle \;\middle|\; \forall i \in \{1, \ldots, n\}, a_i \in A \right\}$.

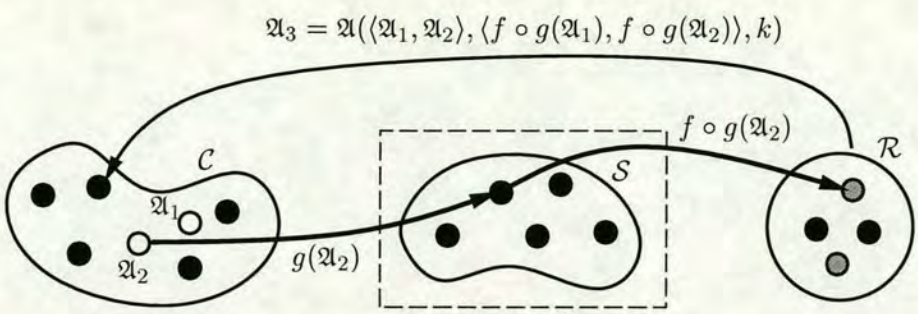


Figure 3.4: The next point in the search is simply obtained by iterating the mapping. Notice that if $f \circ g(\mathfrak{A}_2) = f \circ g(\mathfrak{A}_1)$, the algorithm cannot "know" whether or not it visited a new point in $\mathcal{S}$ (i.e. whether $g(\mathfrak{A}_2) = g(\mathfrak{A}_1)$).

(The integer seed is of course a constant, the same at every application of $\mathfrak{A}'$, and once fixed, $\mathfrak{A}'(k)$ is completely deterministic.) This process is illustrated in figures 3.3 and 3.4.

Because a stochastic search algorithm is perfectly at liberty to ignore the given seed, it will in this sense be taken to include the case of a deterministic search algorithm. Thus, results derived for the stochastic case also apply to the deterministic case.

Note particularly that in both cases, the sequence of points $\langle x_i \rangle$ is *not* assumed to be non-repeating except when this is explicitly stated. For convenience, we will normally focus attention on search algorithms that do not terminate before visiting the entire representation space $\mathcal{C}$, though performance measures will still be at liberty to use only some smaller number of function values (such as the first N).

These definitions of algorithm are certainly broad enough to include all familiar classes of stochastic search algorithms as actually used (e.g. hill climbers, evolutionary algorithms, simulated annealing, tabu search).

3.4 Performance Measures

There are serious methodological problems in devising relevant performance measures that allow different algorithms to be compared on a "fair" basis, whether the algorithms are both from the same class (e.g. both evolutionary algorithms) or from different classes (e.g. a genetic algorithm and a simulated annealing algorithm). In the context of well-defined optimisation problems, we believe that the most sensible framework for comparison is to begin by agreeing a well-defined problem domain over which the comparison is to be performed. Competing algorithms should then be implemented using sample instances from the agreed problem class. The actual comparison between the competing algorithms would then be performed by running the (fixed) programs on a (new) random sample of problems from the class. The hardware used should be the

same for the competing programs, as should be the (wall-clock) time allowed for each algorithm. An agreed performance measure (usually the best solution produced in a fixed time) should be used for comparing the performance on each problem instance.

However, in the field of evolutionary algorithms, the approach suggested above is rarely adopted, and the performance measures used are often bizarre. For example, there has been a peculiar focus on measures such as on-line performance, which measures the average of all points sampled during the course of the run, and off-line performance, which measures the mean performance of the best solution found as a function of time t, each of which is of interest in only a minority of situations (e.g. De Jong, 1975). There has also been a disturbing over-reliance on fixed test suites of problem instances, which inevitably increases the likelihood of artifacts in purported insights into evolutionary methods (Belew, 1992), and little consideration of the relative performance as problem size increases (e.g. see Bilchev & Parmee, 1996).

All the results below will refer rather cavalierly to "arbitrary performance measures", so it will be useful to make clear exactly what the assumed scope of such a performance measure actually is. A performance measure for a search algorithm will be taken to be any measure that depends only on the sequence of objective function values of the images under g of the points in $\mathcal{C}$ chosen by the algorithm. Formally, a performance measure μ will be a function

$$\mu : \mathbb{S}(\mathcal{R}) \longrightarrow \mathcal{M} \tag{3.10}$$

where $\mathcal{M}$ is any set used to measure the "quality" of some sequence of objective function values. Thus, abusing the notation further, we will define the performance of an algorithm $\mathfrak{A}$ on a fitness function f with representation g to be

$$\mu_{f,g}(\mathfrak{A}) \triangleq \mu(\langle f \circ g(\mathfrak{A}_i) \rangle). \tag{3.11}$$

Note that the performance measures defined here depend only on the observed sequence of objective function values, and thus can not distinguish between objective function values deriving from newly visited points in $\mathcal{C}$ and those from points previously visited. This highlights the important distinction between algorithms that (potentially) revisit points in $\mathcal{C}$ and those that never do so. For example, given an algorithm $\mathfrak{A}$ that *does* revisit points in $\mathcal{C}$, we might derive a non-revisiting algorithm $\mathfrak{B}$ that somehow skips any previously visited chromosome, but then clearly $\mathfrak{A}$ and $\mathfrak{B}$ are different algorithms with $\mu_{f,g}(\mathfrak{A}) \not\equiv \mu_{f,g}(\mathfrak{B})$ in general (a fact seemingly overlooked by some authors).

We will also consider the performance of an algorithm over an (unordered) set of functions. By an *overall performance measure* we will mean any measure of the quality of a collection of sequences of objective function values. Given that two different

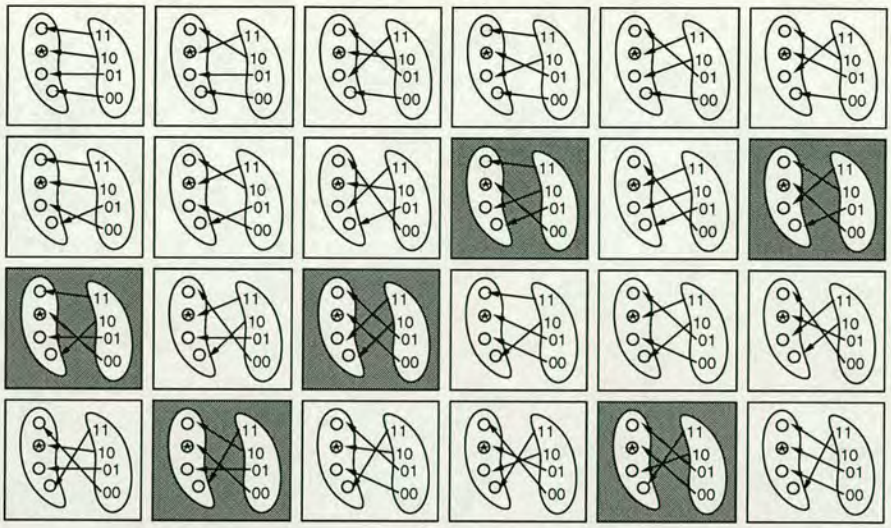


Figure 3.5: This figure shows the 4! invertible mappings between a representation space $\mathcal{C}$ (in this case binary strings) and an arbitrary search space $\mathcal{S}$ of the same size. Note that 1/4 of the mappings (each shown in grey) map the string 00 to the global optimum (shown with a star).

objective functions may generate the same sequence of values, this collection will, in general, be a multiset (bag) of such sequences (i.e. sequences may appear more than once in the collection). Notice also that the collection is taken to be *unordered,* so that an overall performance measure is insensitive to the order in which the functions are presented. A simple example of such an overall performance measure is the average number of function evaluations required to find a global optimum.

3.5 No Free Lunch

3.5.1 Terminology

In order to prove the so-called "No Free Lunch" theorems, we must first introduce some additional terminology:

3.5.1.1 Permutations

Most of the "work" in the following theorems is achieved by using a permutation, i.e. a re-labelling of the elements of one of the spaces (usually $\mathcal{C}$). A permutation of a set A is a relabelling of the elements of the set, i.e. an *invertible* mapping

$$\pi : A \longrightarrow A. \tag{3.12}$$

The set of all permutations of A objects will be denoted $\Pi(A)$. The changes to a faithful representation of a search space $\mathcal{S}$ by a representation space $\mathcal{C}$ under all possible permutations of the elements of $\mathcal{C}$ are shown in figure 3.5.

3.5.1.2 Valid Search Sequence in $\mathcal{S}$

It will sometimes be convenient to refer to a *valid search sequence* in $\mathcal{S}$ in the context of some representation $g \in \mathcal{S}^{\mathcal{C}}_{>}$. A valid search sequence is simply one that can be generated as the image under g of a non-repeating search sequence in $\mathcal{C}$ (perhaps generated by a non-repeating search algorithm $\mathfrak{A}$). For example, if $\mathcal{C}$ contains one more element than $\mathcal{S}$, each (complete) valid search sequence in $\mathcal{S}$ contains one member of $\mathcal{S}$ twice and all others exactly once.

3.5.2 Theorems

The first task in formulating limitations on search is to define what it means for two algorithms to be *isomorphic*. Very simply, the idea is that two algorithms are isomorphic if one can be obtained from the other by a permutation of the representation space. This is what the following definition says.

Definition (Isomorphism for search algorithms). Let $\mathfrak{A}$ and $\mathfrak{B}$ be non-repeating stochastic search algorithms with an objective function $f \in \mathcal{R}^{\mathcal{S}}$ and a growth function $g \in \mathcal{S}^{\mathcal{C}}_{>}$. Then $\mathfrak{A}$ and $\mathfrak{B}$ will be said to be *isomorphic with respect to f and g* if and only if for every pair of seeds $k_{\mathfrak{A}}$ and $k_{\mathfrak{B}}$ there exists a permutation $\pi(k_{\mathfrak{A}}, k_{\mathfrak{B}}) \in \Pi(\mathcal{C})$ such that

$$\langle g(\mathfrak{A}_i) \rangle = \langle g \circ \pi(\mathfrak{B}_i) \rangle \tag{3.13}$$

where $\langle \mathfrak{A}_i \rangle$, $\langle \mathfrak{B}_i \rangle$, are the sequences of chromosomes visited by $\mathfrak{A}$ and $\mathfrak{B}$ when seeded by $k_{\mathfrak{A}}$ and $k_{\mathfrak{B}}$ respectively.

■

The idea of isomorphism between search algorithms is important because the following results will demonstrate that isomorphic algorithms perform equally in some extremely strong senses. Intuitively, two algorithms which visit the same sequence in $\mathcal{S}$ must necessarily observe the same sequence of function values in $\mathcal{R}$, so share the same performance:

$$\begin{aligned} \langle g(\mathfrak{A}_i) \rangle &= \langle g \circ \pi(\mathfrak{B}_i) \rangle \\ \Longrightarrow \langle f \circ g(\mathfrak{A}_i) \rangle &= \langle f \circ g \circ \pi(\mathfrak{B}_i) \rangle \\ \Longrightarrow \mu_{f,g}(\mathfrak{A}) &= \mu_{f,g \circ \pi}(\mathfrak{B}). \end{aligned} \tag{3.14}$$

Theorem (Isomorphism for non-repeating algorithms). *Let $\mathfrak{A}$ and $\mathfrak{B}$ be non-repeating stochastic search algorithms using the same objective function f and the same representation space $\mathcal{C}$. Then for every representation $g \in \mathcal{S}^{\mathcal{C}}_{>}$, $\mathfrak{A}$ and $\mathfrak{B}$ are isomorphic with respect to f and g.*

Proof. Let g, g' be two different growth functions in $\mathcal{S}^{\mathcal{C}}_{>}$, and consider any fixed pair of seeds $k_{\mathfrak{A}}$, $k_{\mathfrak{B}}$ so that we may treat $\mathfrak{A}$ and $\mathfrak{B}$ as deterministic algorithms. Clearly, $\mathfrak{A}$ must visit a different sequence of points in $\mathcal{S}$ under g and g'. (This may be clarified by looking again at figures 3.3 and 3.4. Since the first point, $\mathfrak{A}_1$, chosen by $\mathfrak{A}$ is independent of the growth function, it is easy to see inductively that if $\mathfrak{A}$ visited the same sequence of points in $\mathcal{S}$ under g and g', then g and g' would in fact be equal, as every point in $\mathcal{C}$ will eventually be visited by a non-terminating, non-repeating algorithm.) This shows that any non-revisiting algorithm generates every valid search sequence in $\mathcal{S}$ for some $g \in \mathcal{S}^{\mathcal{C}}_{>}$, and that any such search sequence determines g uniquely (for the algorithm $\mathfrak{A}$). Conversely each growth function g defines a unique search sequence in $\mathcal{S}$, namely $\langle g(\mathfrak{A}_i) \rangle$.

It is thus clear that for every $g_{\mathfrak{A}} \in \mathcal{S}^{\mathcal{C}}_{>}$, yielding search sequence $\langle g_{\mathfrak{A}}(\mathfrak{A}_i) \rangle$, there exists a $g_{\mathfrak{B}} \in \mathcal{S}^{\mathcal{C}}_{>}$ such that $\langle g_{\mathfrak{B}}(\mathfrak{B}_i) \rangle = \langle g_{\mathfrak{A}}(\mathfrak{A}_i) \rangle$. Further, it is easy to see that some permutation $\pi \in \Pi(\mathcal{C})$ exists such that $g_{\mathfrak{B}} = g_{\mathfrak{A}} \circ \pi$—specifically, $\pi(\mathfrak{B}_i) \triangleq \mathfrak{A}_i$, which is a permutation of $\mathcal{C}$ since both $\langle \mathfrak{A}_i \rangle$ and $\langle \mathfrak{B}_i \rangle$ are non-repeating enumerations of $\mathcal{C}$. This is exactly the permutation required to demonstrate isomorphism of $\mathfrak{A}$ and $\mathfrak{B}$ with respect to f and g. □

Theorem (Isomorphic algorithms perform equally over permutations of $\mathcal{C}$). *Let $\mathfrak{A}$ and $\mathfrak{B}$ be stochastic search algorithms that are isomorphic with respect to some objective function $f \in \mathcal{R}^{\mathcal{S}}$ and some set of growth functions $\mathcal{G} \subset \mathcal{S}^{\mathcal{C}}_{>}$ that is closed under permutation of $\mathcal{C}$. Then their overall performance on the ensemble of search problems defined by all growth functions in $\mathcal{G}$ is identical, regardless of the overall performance measure chosen.*

Proof. For any valid search sequence $\langle g(\mathfrak{A}_i) \rangle \in \mathbb{S}(\mathcal{S})$, we can find a permutation $\pi \in \Pi(\mathcal{C})$ such that $\langle g \circ \pi(\mathfrak{B}_i) \rangle = \langle g(\mathfrak{A}_i) \rangle$ from the definition of isomorphism. By symmetry, the converse is also true, so every search sequence in $\mathcal{S}$ generated by either algorithm is also generated by the other. Since it was shown in the proof of the previous theorem that different growth functions define different search sequences for a given algorithm, this suffices to show that the set of search sequences in $\mathcal{S}$ generated by $\mathcal{G}$ is the same for the two algorithms, so their performance over $\mathcal{G}$ must be equal. □

Perhaps the most interesting consequence of this theorem is that if the entire ensemble of representations in $\mathcal{S}^{\mathcal{C}}_{>}$ is considered, the only factor that differentiates algorithms is the frequency with which they revisit points. This is because, over this ensemble of representations, visiting one new point is no different from visiting any other.

Corollary (Overall Performance Measures). *Consider search algorithms in which the representation space $\mathcal{C}$ is the same as the search space $\mathcal{S}$. The overall performance of*

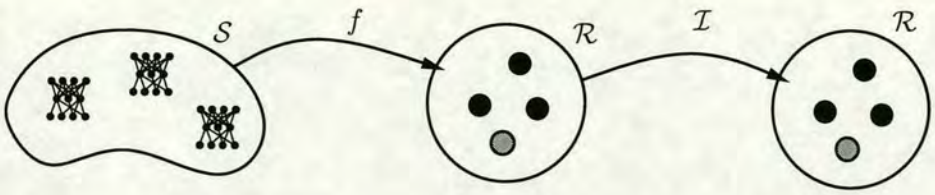


Figure 3.6: The relabelling above, with $\mathcal{I}$ as the identity mapping, establishes that all isomorphic algorithms have identical overall performance over all functions in $\mathcal{R}^{\mathcal{S}}$ against any overall performance measure. If f is not surjective, the problem has to be reformulated using the image $f(\mathcal{S})$ in place of $\mathcal{R}$ for the theorem to apply directly, but since the theorem applies to each such image, it clearly also applies to their union. A further corollary to this is the "No Free Lunch" Theorem (Wolpert & Macready, 1995).

all search algorithms that do not revisit points in $\mathcal{C}$ on the ensemble of search problems defined by all objective functions in $\mathcal{R}^{\mathcal{S}}$ is identical, regardless of the overall performance measure chosen.

■

Proof. For all functions in $\mathcal{R}^{\mathcal{S}}_{>}$, the result follows directly from the theorems by a trivial relabelling (figure 3.6) where we take $\mathcal{S} = \mathcal{R}$, f to be the identity mapping and $\mathcal{G}$ to be the set of all surjective functions from $\mathcal{S}$ to $\mathcal{R}$. Similarly, for any set of functions surjective onto a subset of $\mathcal{R}$, the theorem still clearly holds taking $\mathcal{R}$ as the given subset of $\mathcal{R}$. Since the set of all functions in $\mathcal{R}^{\mathcal{S}}$ is clearly just the union over such surjective subsets, we have the required result.

■

Corollary (The No Free Lunch Theorem). *The mean performance of all isomorphic search algorithms is identical over the set of all functions in $\mathcal{R}^{\mathcal{S}}$ for any chosen performance measure.*

Proof. This follows immediately from the previous corollary. □

Wolpert & Macready (1995) conjecture that there may be what they term "minimax" distinctions between pairs of algorithms. Here they have in mind "head-to-head" comparisons on a "function-by-function" basis, where, for example, one algorithm ($\mathfrak{A}$) might outperform another ($\mathfrak{B}$) on some performance measure more often than the reverse is the case. This is clearly true, as table 3.1 shows. Notice, however, that such comparisons may be non-transitive, i.e. given a third algorithm $\mathfrak{C}$, it might be that case that

$$(\mathfrak{C} > \mathfrak{B} \text{ and } \mathfrak{B} > \mathfrak{A}) \not\Rightarrow \mathfrak{C} > \mathfrak{A}, \tag{3.15}$$

where $>$ means "outperforms" in the minimax sense described above. An example of exactly this situation, where $\mathfrak{C} > \mathfrak{B}$ and $\mathfrak{B} > \mathfrak{A}$ but $\mathfrak{A} > \mathfrak{C}$ is also shown in the table.

Function $f((1,2,3))$	Time to Minimum $\mathfrak{A}$	$\mathfrak{B}$	$\mathfrak{C}$	Winner $\mathfrak{A}$ vs. $\mathfrak{B}$	Winner $\mathfrak{B}$ vs. $\mathfrak{C}$	Winner $\mathfrak{C}$ vs. $\mathfrak{A}$
(0,0,0)	1	1	1	tie	tie	tie
(0,0,1)	1	1	2	tie	$\mathfrak{B}$	$\mathfrak{A}$
(0,1,0)	1	2	1	$\mathfrak{A}$	$\mathfrak{C}$	tie
(0,1,1)	1	3	2	$\mathfrak{A}$	$\mathfrak{C}$	$\mathfrak{A}$
(1,0,0)	2	1	1	$\mathfrak{B}$	tie	$\mathfrak{C}$
(1,0,1)	2	1	3	$\mathfrak{B}$	$\mathfrak{B}$	$\mathfrak{A}$
(1,1,0)	3	2	1	$\mathfrak{B}$	$\mathfrak{C}$	$\mathfrak{C}$
(1,1,1)	1	1	1	tie	tie	tie
Overall winner				$\mathfrak{B}$	$\mathfrak{C}$	$\mathfrak{A}$

Table 3.1: Consider all functions from $\mathcal{S} = \{1,2,3\}$ to $\mathcal{R} = \{0,1\}$ and three search algorithms $\mathfrak{A}, \mathfrak{B}$ and $\mathfrak{C}$, each of which simply enumerates $\mathcal{S}$. In particular, take $\langle \mathfrak{A}_i \rangle = \langle 1,2,3 \rangle$, $\langle \mathfrak{B}_i \rangle = \langle 2,3,1 \rangle$ and $\langle \mathfrak{C}_i \rangle = \langle 3,1,2 \rangle$. The table lists all eight functions in $\mathcal{R}^{\mathcal{S}}$ and compares the algorithms pairwise with respect to the particular performance measure 'number of steps to find a global minimum'. This illustrates the "minimax" distinction conjectured by Wolpert & Macready (1995), but also shows that this is non-transitive. Indeed, the overall performance of each algorithm is identical, regardless of what overall performance measure is chosen. (Columns 2, 3 and 4 are permutations of each other, illustrating this.)

Corollary (Enumeration). *No non-repeating search algorithm can outperform enumeration over the ensemble of problems defined by* $\mathcal{R}^{\mathcal{S}}$.

Proof. This is also immediate from the theorem, noting that enumeration is a non-revisiting search algorithm. □

3.6 Polynomial-time No Free Lunch

One potential argument against the results of the preceding section is that they do not relate to "practical" algorithms, implemented on real digital computers using limited time and space resources. A large literature on the theory of polynomial-time computation addresses such issues (for example, see the widely respected Garey & Johnson, 1979). As shown below, it is possible to recast the No Free Lunch results in this form, demonstrating the practical relevance of the results.

3.6.1 Polynomial-time Algorithms

Informally, a polynomial-time algorithm for a particular problem domain is one which can be computed in a number of steps bounded above by a function which is polynomial in the size of the problem instance to which it is applied. Using the example of the travelling sales-rep problem, consider the domain of n-city TSPs with (bounded) integral

inter-city distances. The size of any particular instance of this problem is polynomial in n (e.g. a list of $n(n-1)/2$ edge lengths, or a list of n pairs of city coordinates given a distance metric), so a polynomial-time algorithm would be one which terminated within $q(n)$ steps for a polynomial function q.

For the kinds of "difficult" search problems that interest us, another convenient metric for the size of a problem instance is the size of a candidate solution to that problem. (Clearly if the size of a solution were not polynomially bounded in the size of the problem instance then no algorithm could even write any solution in polynomial time, and conversely, if the size of a problem instance were not polynomially bounded in the size of a candidate solution then an effective sub-polynomial search strategy would simply be to enumerate all candidate solutions.) Returning to the example of the TSP, a candidate solution (a tour) can be encoded in space of order $n \log n$ (a list of n integers between 1 and n). Clearly either of $n \log n$ and n^2 can be polynomially bounded by the other.

Assume then that we can identify any specific member of $\mathcal{C}$ using $l = \log |\mathcal{C}|$ bits; that is, $|\mathcal{C}| = O(2^l)$. (In the terms of Garey & Johnson, 1979, this simply means that we have a *concise* encoding for $\mathcal{C}$.) Thus we are interested in search algorithms that are polynomial in l, and must examine the implications of this requirement.

From our definition, a search algorithm $\mathfrak{A}$ is allowed access to the sequences $\langle \mathfrak{A}_i \rangle_{i=1}^n$ and $\langle f \circ g(\mathfrak{A}_i) \rangle_{i=1}^n$ at its $n+1$st step. If the algorithm is to make any use of these sequences at all and execute in polynomial time (in l), it is clear that both $g(\cdot)$ and $f(\cdot)$ must be calculable in polynomial time. Because each step of the search algorithm $\mathfrak{A}$ takes at least $O(l)$ time simply to write its output $\mathfrak{A}_i$, in order for $\mathfrak{A}$ to run in polynomial time we must limit it to a number of steps which is also polynomial in l. (Intuitively, any practical algorithm for any reasonable problem can only explore a vanishingly small fraction of the search space.)

3.6.2 Search Algorithms that Revisit Points

Before considering the "No Free Lunch" theorem for polynomial algorithms it is first useful to consider the implications of the restrictions of the earlier theorems to non-revisiting searches. In the form that they state the No Free Lunch Theorem, Wolpert & Macready only consider algorithms that do not revisit points already considered, and seem to regard the issue of revisiting as a trivial technicality. In fact, most searches do not have this property, and it is worth noting that there are good reasons for this. Perhaps the most obvious is finite memory of real computers, but this is not the most important explanation. For while there are undoubtedly searches carried out for which it would be impossible today to store every point visited, as main memory sizes increase, this becomes ever less of an issue. (At the very least it is typically feasible to store a

much larger history than even most population-based algorithms choose to do.) Much more significant is the time needed to process large amounts of data—even merely to perform a check against every point so far evaluated requires, at best, log time. Depending on the details of the algorithm, other functions may take even longer. As an example, ranking schemes used in some genetic algorithms (Baker, 1987) require their populations to be sorted.

Because most algorithms used do revisit points, there is the potential, in principle, to improve real search techniques without reference to the particular problem being tackled. It should be clear that the kinds of performance measures relevant in this case concern the average number of points sampled, *including* points visited multiple times, to reach a solution of a certain quality, or the total "wall-clock" time to achieve the same goal. (Indeed, this "best-so-far" graph as a function of the number of function evaluations is precisely the graph shown in most empirical papers on stochastic search.) Thus, for example, when Whitley (1989) and Davis (1991b) advocate barring duplicates in the population of a genetic algorithm, it is perfectly possible that this will indeed prove beneficial over an entire ensemble of problems $\mathcal{R}^{\mathcal{S}}$ by reducing the likelihood of revisiting. The suggested benefits of enforcing uniqueness of solutions are that it leads to more accurate sampling (with respect to target sampling rates, Grefenstette & Baker, 1989), increases diversity in the population and makes better use of the limited "memory" that the population represents. On the other hand, there is a clear cost associated with maintaining uniqueness, in that it requires checking each new (tentative) solution generated against the current population. The point here is not *whether* enforcing uniqueness *does* improve evolutionary search in general, but that in principal it *could* do so. The same applies to any change that affects diversity in a population-based search, and which is therefore likely to affect the frequency with which points are revisited.

Note that the distinction between revisiting and non-revisiting algorithms is particularly important when considering practical algorithms. Although it is simple to generate a "non-revisiting completion" of an algorithm $\mathfrak{A}$, it is not clear that such a completion can preserve the polynomial nature of $\mathfrak{A}$. For instance, we could define a non-revisiting algorithm $\mathfrak{B}$ from $\mathfrak{A}$ by simply skipping any previously visited point that $\mathfrak{A}$ would otherwise have generated:

$$\mathfrak{B}_i \triangleq \mathfrak{A}_{k_i}$$
$$\text{where } k_1 = 1, k_{i+1} = \min\{j \in \mathbb{Z}^+ \mid \mathfrak{A}_j \notin \{\mathfrak{A}_1, \ldots, \mathfrak{A}_{k_i}\}\} \tag{3.16}$$

(To make a standard genetic algorithm non-revisiting would involve maintaining a lookup table of every previously evaluated chromosome, so that the objective function need only be invoked for new chromosomes—a form of pan-generational uniqueness.)

However, the k_i may increase faster than polynomially, so that, for instance, applying this technique to a simple genetic algorithm might not result in a polynomial algorithm.

An alternative method for generating a non-revisiting algorithm from a revisiting one would be to sample a random unvisited point whenever the original algorithm would generate a previously visited point (this could be accomplished by repeatedly generating points at random until an unvisited one was generated). The resulting algorithm would, in expectation, run in polynomial time since most points will necessarily be unvisited after a polynomial number of steps. However, this would tend to result in a non-revisiting search strategy which degenerates to random search, particularly as the underlying algorithm nears "convergence" (i.e. as it becomes increasingly likely to generate previously visited points).

Although "real" (polynomial-time) algorithms may not be non-revisiting, it may still possible to make "overall" comparisons between them, but only in the negative sense that one is less bad relative to enumeration than another.

3.6.3 No Free Lunch Theorems

By restricting attention to representations g which are polynomial, our previous theorem on the equal performance of isomorphic algorithms can no longer be applied. This is so because the permuted representations $g \circ \pi$ used in that argument may not be polynomial—in general π is a complex mapping of $\mathcal{C} \longrightarrow \mathcal{C}$ which is not necessarily polynomial-time in $\log |\mathcal{C}|$. This has the result that the isomorphism constructed previously may identify an algorithm using a polynomial representation with another using a non-polynomial one (recall that for an algorithm to execute in polynomial time we require that g, f be polynomial time).

Theorem (Equal performance over polynomial representations of $\mathcal{C}$). *Let $\mathfrak{A}$ and $\mathfrak{B}$ be non-repeating polynomial-time algorithms (limited to a polynomial number of steps) that are isomorphic with respect to some objective function $f \in \mathcal{R}^{\mathcal{S}}$ and some set of polynomial growth functions $\mathcal{G} \subset \mathcal{S}^{\mathcal{C}}_{>}$ that is closed under polynomial permutations of $\mathcal{C}$. Then their overall performance on the ensemble of search problems defined by all growth functions in $\mathcal{G}$ is identical, regardless of the overall performance measure chosen.*

Proof. Consider any fixed search sequence $\langle s_i \rangle_{i=1}^{n} \in \mathbb{S}(\mathcal{S})$, and let $\mathcal{G}_{\mathfrak{A}} \subset \mathcal{G}$ be the set of all growth functions for which $\mathfrak{A}$ generates that search sequence; that is, $g \in \mathcal{G}_{\mathfrak{A}} \Longrightarrow \langle g(\mathfrak{A}_i) \rangle_{i=1}^{n} = \langle s_i \rangle_{i=1}^{n}$; and let $\mathcal{G}_{\mathfrak{B}}$ be defined similarly for $\mathfrak{B}$.

We contend that $|\mathcal{G}_{\mathfrak{A}}| = |\mathcal{G}_{\mathfrak{B}}|$. Note first that regardless of the growth function in $\mathcal{G}_{\mathfrak{A}}$, the sequence of chromosomes $\langle \mathfrak{A}_i \rangle_{i=1}^{n} \in \mathbb{S}(\mathcal{C})$ visited by $\mathfrak{A}$ must be fixed (exactly as argued in the previous proof of algorithmic isomorphism). Similarly, all growth functions in $\mathcal{G}_{\mathfrak{B}}$ lead to a fixed sequence $\langle \mathfrak{B}_i \rangle_{i=1}^{n}$ of chromosomes.

Proceeding in a similar way to our previous argument, we will explicitly construct a polynomial-time permutation π which can be used to map growth functions in $\mathcal{G}_{\mathfrak{A}}$ to functions in $\mathcal{G}_{\mathfrak{B}}$, allowing us to show that the two sets are of equal size.

We define $\pi : \mathcal{C} \longrightarrow \mathcal{C}$ by:

$$\pi(x) = \begin{cases} \mathfrak{A}_i, & x = \mathfrak{B}_i; \\ (\langle \mathfrak{B}_i \rangle_{i=1}^{n} \setminus \langle \mathfrak{A}_i \rangle_{i=1}^{n})[j], & x = (\langle \mathfrak{A}_i \rangle_{i=1}^{n} \setminus \langle \mathfrak{B}_i \rangle_{i=1}^{n})[j]; \\ x, & \text{otherwise.} \end{cases} \tag{3.17}$$

where $\langle x_i \rangle_{i=1}^{n} \setminus \langle y_i \rangle_{i=1}^{n}$ is the sequence obtained by deleting all those elements from the the former sequence which appear in the latter, and $\langle x_i \rangle[j] \triangleq x_j$ (i.e. the jth element of the sequence).

Now, π is a permutation of $\mathcal{C}$ (as we can trivially construct its inverse), and for n a polynomial function of $l = \log |\mathcal{C}|$, π can be evaluated in polynomial time. Further, $g \in \mathcal{G}_{\mathfrak{A}} \Longrightarrow g \circ \pi \in \mathcal{G}_{\mathfrak{B}}$ (by construction), and $g \circ \pi$ is polynomial as both g and π are polynomial. Finally, for any pair $g_1 \neq g_2 \in \mathcal{G}_{\mathfrak{A}}$, $g_1 \circ \pi \neq g_2 \circ \pi$ (as $g_1 \neq g_2 \in \mathcal{G}_{\mathfrak{A}} \Longrightarrow \exists x$ s.t. $g_1(x) \neq g_2(x)$, so that taking $y = \pi^{-1}x$, $g_1 \circ \pi(y) \neq g_2 \circ \pi(y) \Longrightarrow g_1 \circ \pi \neq g_2 \circ \pi$). Thus, from every unique element of $\mathcal{G}_{\mathfrak{A}}$, we can construct a unique element of $\mathcal{G}_{\mathfrak{B}}$, so that we must have $|\mathcal{G}_{\mathfrak{A}}| \leq |\mathcal{G}_{\mathfrak{B}}|$. By an exactly analogous argument, we see that $|\mathcal{G}_{\mathfrak{B}}| \leq |\mathcal{G}_{\mathfrak{A}}|$, so that in fact we must have $|\mathcal{G}_{\mathfrak{A}}| = |\mathcal{G}_{\mathfrak{B}}|$.

By considering each possible search sequence of the fixed allowable number of steps, we see that the set of search sequences which the two algorithms produce from the representations in $\mathcal{G}$ are identical, so that their performance must be equal. □

3.7 Implications for Search

The primary implication of the no free lunch theorems is that there is no universal "good" algorithm. Thus it is futile to seek such a "silver bullet" algorithm, or to attempt to prove strong generic results concerning the performance of search algorithms. NFL tells us that true "black-box" optimisation can be at most as efficient as enumerative search, despite the claims of some authors, e.g. Kargupta, 1995. This includes all forms of learning algorithms (Schmidhuber, 1995), evolving representations (e.g. delta-coding, Whitley *et al.*, 1991a; dynamic parameter encoding, Schraudolph & Belew, 1990; etc.).

Furthermore, we conclude that all such attempts are necessarily hopeless *unless* they are placed firmly within the context of a well-specified (and thus limited) search domain. Fundamentally, this means that we can only hope to create useful search algorithms by somehow incorporating knowledge of the problem domain into the algorithm. Several routes might be envisioned for doing this. Most obvious is to design problem-specific heuristics, and indeed a large literature exists here (see for example Pearl, 1984). Is there no hope then for generic algorithms? Although this might seem to be the case

in light of the arguments above, things are not necessarily so bleak. The approach we advocate here is to design a generic algorithm that can be tailored in a formulaic way when attacking any new problem.

The formulation presented in chapter 4 postulates a problem-dependent characterisation χ that captures knowledge about a problem domain. This characterisation mechanically generates a *formal representation* (representation space and growth function) for any instance of the problem, as shown in the bottom part of figure 4.1.

Using this idea of the representation as a fulcrum, we are able to split a practical search strategy into a completely generic yet precisely defined algorithm which acts via the problem-specific representation. In this way, all domain knowledge is explicitly captured within the representation, and the algorithm simply manipulates solutions via that representation in a well-defined way.

This approach has several advantages. First, we are able to specify precisely problem-independent search algorithms that can be transferred between problem domains. Second, focusing on the choice of representation as the vehicle for incorporating domain knowledge in the search process highlights the importance of this choice. The powerful and general formalism we develop allows us to *characterise* the search domain; stating mathematically our beliefs about how performance relates to features of the candidate solutions. From this mathematical characterisation of our specialised problem knowledge, we can formulaically derive a representation for any problem instance within the domain of interest.

The general goal here is to forge a strong link between explicitly stated beliefs about which features of a search domain affect performance and the quality of the instantiated search algorithm. In particular, the aim is that good characterisations and beliefs lead to good search performance, but equally importantly that poor characterisations result in poor search.

The move operators used to construct a search algorithm can then be defined on the representation space, and the quality of any chromosome can be determined using the growth function in conjunction with the fitness function. One obvious method for achieving the goal of specifying search in a problem-independent way is to fix the representation space, and then to choose an appropriate growth function for each new problem domain. Such a search algorithm would sample chromosomes from the fixed representation space, using the growth and fitness functions as a "black-box" to evaluate them. This idea is addressed in more detail in section 3.7.1. However, we argue that only by abstracting the definition of a search algorithm away from a fixed representation-space (just as we strove for independence from a particular problem domain), can we realise the goal of a truly problem-independent algorithm while at the same time making the rôle played by domain knowledge properly explicit.

3.7.1 Canonical (Fixed) Representations

The traditional genetic algorithm (e.g. the Simple Genetic Algorithm of Goldberg, 1989c) is defined over a fixed representation space, namely that of binary strings. A common perception is that to employ such an algorithm for a new problem, one need only define a fitness function. (Indeed, standard software packages exist which literally require only computer code for a fitness function, e.g. GENESIS; Grefenstette, 1984.) For problems defined explicitly over binary strings (one counting, royal road, etc.) this does not present any difficulty. For others, such as real-parameter optimisation, some encoding from the problem variables into binary strings must be formulated, in order that the fitness of binary chromosomes can then be calculated by decoding them. However, such "shoe-horning" may make much of the structure of the search problem unavailable to the algorithm in terms of heritable allele patterns (see for example the discussion of meaningful alphabets in Goldberg, 1990). For problems in which candidate solutions are more complicated objects (e.g. the travelling sales-rep problem) a direct binary encoding may be unnatural or even infeasible. A particular case is when the "natural variables" of the problem are not orthogonal, in that the valid settings of one variable depend on the value of another (e.g. permutations). Faced with such a situation, practitioners typically adopt an *ad hoc* approach, drawing on evolutionary "concepts" to define pragmatic new move operators perceived as appropriate in the new domain (e.g. operators such as sub-tour inversion, partially matched crossover and order crossover have been devised for the TSP; Oliver *et al.*, 1987).

Although the use of a fixed representation space is widespread, it is unsatisfactory for a number of reasons. Algorithms based on this idea fail to be truly independent of problem (particularly for problems with naturally non-orthogonal representations). As such, it is not possible to transfer a given algorithm to an arbitrary problem domain. Because such algorithms have only limited domains of applicability, it becomes difficult to make meaningful comparisons between different algorithms. (Attempting to compare, for instance, "genetic algorithms" and "simulated annealing" is futile until both a problem domain and set of move operators are specified, to define each "algorithm" precisely.) Finally, a fixed representation space makes it much more difficult to incorporate knowledge about the structure of the problem—one is forced to change either the growth function (the "genotype-phenotype" mapping) or the move operators, making the algorithm even less problem-independent.

3.7.2 Representation and Algorithm Duality

Consider two different algorithms $\mathfrak{A}$ and $\mathfrak{B}$, operating on the same problem f from $\mathcal{R}^{\mathcal{S}}$ and using the same representation space $\mathcal{C}$, but with different growth functions $g, g' \in \mathcal{S}^{\mathcal{C}}$. It is clear that there are some situations in which even though $\mathfrak{A} \neq \mathfrak{B}$, and

$g \neq g'$, the differences "cancel out" so that they perform the same search. (Indeed, an obvious case arises when the algorithms are isomorphic, and $g' = g \circ \pi$ for some suitable permutation π.) This might lead us to suspect that there is a *duality* between representations and algorithms, so that any change that can be effected by altering the representation could equally be achieved by a corresponding alteration to the algorithm, and any change made to the algorithm could equally be effected by a corresponding change to the representation. In fact, it has been shown (Radcliffe, 1994) that while the first of these statements is true, the second is not, in the general case of algorithms that are allowed to revisit points, or if the stochastic element is not "factored out" by fixing the seed. The duality—if that it be—is only partial.

If, for example, we consider a canonical genetic algorithm (say the Simple Genetic Algorithm—Goldberg, 1989c), but allow either one-point or uniform crossover, then there is clearly no change of representation that transforms one of these algorithms into the other, when revisiting effects are considered—either over all possible seeds for the random number generator, or even for a particular seed. (For example, consider an algorithm generating a large number of children from a population of binary chromosomes which happens to contain only strings of the form $000\cdots0$ and $111\cdots1$. With uniform crossover and the right sequence of random numbers, any solution can be generated, while with one-point crossover only a fixed subset of form $000\cdots111$ and $111\cdots000$ will appear, repeatedly.) Moreover, there is no reason to assume that their performance over the class of all representations in $\mathcal{S}^{\mathcal{C}}_{>}$ (or indeed, all objective functions $f \in \mathcal{R}^{\mathcal{S}}$) will be the same. In terms of the theorems described earlier, this merely amounts to the observation that revisiting algorithms are not, in general, isomorphic. However, this formally trivial statement has rather strong practical implications given that, as noted earlier, establishing whether a point has been previously sampled requires significant computational effort.

Despite the fact that more power is available by changing the algorithm than from merely changing the representation, the clear separation between representation and search algorithm remains extremely valuable, as will be explored in chapters 4 and 5.

3.7.3 Restricted Problem Classes

One of Wolpert and Macready's central messages, with which we agree strongly, is that if algorithms are to outperform random search, they must be matched to the search problem at hand. ("If no domain-specific knowledge is used in selecting an appropriate representation, the algorithm will have no opportunity to exceed the performance of an enumerative search"—Radcliffe, 1994.)

What is required is a methodology by which both good representations and good algorithms can be designed and recognised for particular problem classes. In the case

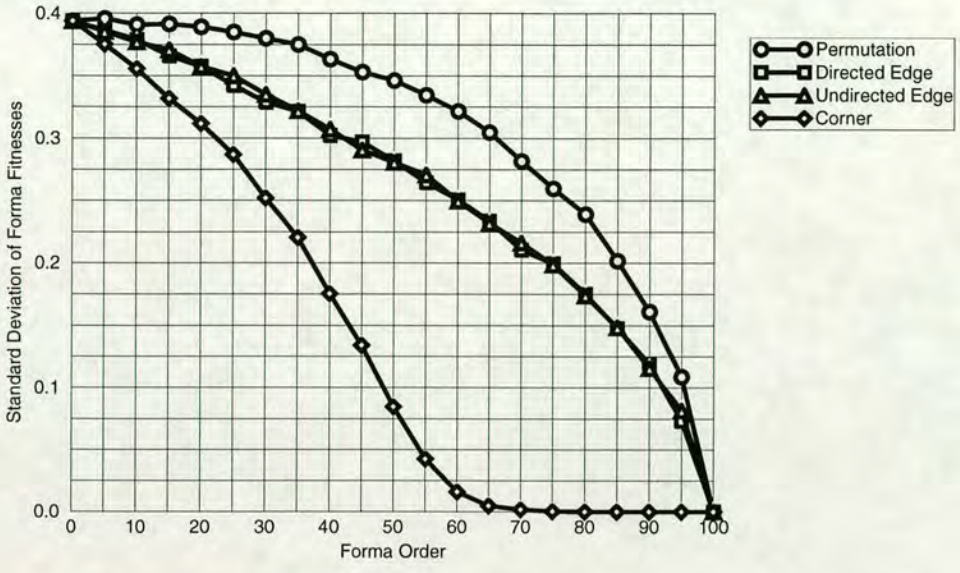


Figure 3.7: The graph shows the mean standard deviation of tour length as a function of schema (forma) order for each of four representations of the TSP. These particular results are from a single problem instance generated using 100 samples drawn from 100 randomly generated schemata at each order shown, though the graph would look nearly identical if averaged sensibly over all 100-city TSPs.

of representations, some preliminary steps have been taken, including work by Davidor (1990), Kauffman (1993) and Radcliffe & Surry (1994b), and from a different perspective, there is an extensive literature on deception, a particular form of linear non-separability (Goldberg, 1989a, 1989b; Whitley, 1991; Das & Whitley, 1991; Grefenstette, 1992; Louis & Rawlins, 1993). All of these examine some form of questions about what precisely "some degree of linear separability" means, how it can be measured and how it may be exploited by algorithms.

One approach that seems particularly fruitful is to measure properties of representations over well-defined problem classes, and then to seek to construct a theory of how such measured properties can be exploited by algorithms. In chapter 7 (based on Radcliffe & Surry, 1994b), we measure the variance of objective function value for four different representations for the travelling sales-rep problem (TSP) as a function of schema order to obtain the graph shown in figure 3.7. (A schema—or forma—fixes certain components of a member of $\mathcal{C}$, with order measuring the number of such fixed components.) The lower variance for the "corner" representation indicates that objective function value is more nearly linearly separable with respect to the components of this representation than is the case with the other representations. Although the four representations are all highly constrained, well-defined move operators were specified solely in terms of manipulations of $\mathcal{C}$: this is what is meant by representation-

independent move operators. We were therefore able to compare the performance of identical algorithms with these four representations over a range of problem instances. The results, as expected, indicated that on standard performance metrics, most of the algorithms used worked better with the lower variance representations than their higher variance counterparts. This suggests strongly that developing further measurements of representation quality, together with further ways of characterising representation-independent operators (such as the notions of respect, assortment and transmission, developed by Radcliffe, 1991a, 1994) and a theory linking those together, is an urgent task. Preliminary steps towards this goal are discussed in chapter 6.

3.8 Discussion

In this chapter we have provided a formal demonstration of various fundamental limitations on search algorithms, with particular reference to evolutionary algorithms. These results establish clearly the central rôle of representation in search, and point to the importance of developing a methodology for formalising, expressing and incorporating domain knowledge into representations and operators, together with a theory to underpin this.

The results demonstrate the futility of trying to construct universal algorithms, or universal representations. For example, neither "binary" nor gray coding (Caruana & Schaffer, 1988) can be said to be superior, either to each other or to any other representation, without reference to a particular problem domain. One immediate practical consequence of this should be a change of methodology regarding test suites of problems and comparative studies. Rather than developing a fixed (small) suite of problems for refining algorithms and representations, we believe that a significantly more useful approach consists of developing algorithms for a well-specified *class* of problems. The idea would be to refine algorithms using any small, randomly chosen subset of problems from this class, but to compare performance only against different randomly selected problem instances from the same class. Of particular interest is the comparative performance with respect to problem size; i.e. on larger and larger instances from the problem domain. We believe that a move towards this philosophy of comparative studies would allow the development of much more systematic insights.

Chapter 4

Formal Representations

The principle of strategy is having one thing, to know ten thousand things.
(Miyamoto Musashi)

The "No Free Lunch" theorems presented in chapter 3 show that it is futile to search for universally efficient algorithms. Section 3.7 showed that any effective search algorithm must incorporate some problem-specific bias based on domain knowledge. It was argued that the representation of the search domain provides a natural mechanism for doing so.

By searching over a set of structures (chromosomes) representing the objects in the search space rather than directly over the objects themselves we are able both to employ template search algorithms defined independently of any specific problem domain, and to make explicit our beliefs about the relationship between fitness and solution structure. This begins to support testable hypotheses about performance of alternative algorithms and representations.

In this chapter we propose a methodology which allows us to derive formulaically a representation for any problem domain of interest. This is based on *forma analysis* (Radcliffe, 1991a, 1991b, 1994), in which a solution is represented by its pattern of membership in specified subsets of the search space. These subsets are called *formae* (generalised schemata), and it is asserted that they capture beliefs about problem structure—in particular, that they group solutions of related performance—and that the objective function is to *some* degree separable over them. As explored in chapter 5, it is then possible to define "representation-independent" operators which manipulate the forma membership properties of solutions so as to reflect our beliefs about the problem structure, resulting in effective search algorithms.

In section 4.2 we postulate a problem-dependent characterisation χ that captures beliefs about problem structure, and which can be used mechanically to generate a *formal representation* (representation space and growth function) for any instance of the problem, by defining a number of equivalences over the search space. These equivalences induce the formae required to represent and manipulate solutions in formal algorithms.

The properties of such representations are explored in section 4.3, and in section 4.4 we discuss the implications of this methodology for deriving problem-specific search strategies.

4.1 Representation in Search

As argued in chapter 3, it is important to introduce the notion of representation when tackling a search problem. Use of a representation (made up of a representation space and an associated growth function; see figure 3.2) allows us to search in an abstract space of chromosomes (which we can define for any problem), rather than in the problem-specific space of solutions. Note that we focus here on formal mathematical representations, which determine the (indirect) actions of the genetic operators in the search space, rather than the physical representation of solutions in (say) a digital computer. This important but potentially confusing distinction is addressed in section 4.1.1.

Using a (formal) representation dissociates the search algorithm from the problem domain, and provides an explicit mechanism by which to capture problem-specific beliefs about the relationship between solution structure and fitness. This chapter demonstrates that a general method for defining a representation is to classify subsets of solutions according to characteristics which they share.

In his seminal work, Holland (1975) proceeded using exactly this approach. He identified subsets of a search space of binary strings using *schemata*—sets of strings that share particular bit values. His Schema Theorem shows how the *observed* fitness of any schema in a population can be used to bound the *expected* instantiation of the same schema in the next generation, under the action of fitness-proportionate selection. Several authors then generalised the notion of a schema and have shown that the theorem applies to arbitrary subsets of the search space, provided that suitable disruption coefficients are chosen (Radcliffe, 1991a; Vose & Liepins, 1991b).

In particular, Radcliffe (1991a, 1991b) has developed the idea of *forma analysis,* in which general subsets of the search space are termed *formae*. Typically, the formae are defined as the equivalence classes induced by a set of equivalence relations, although this need not be the case. Any solution can then be identified by specifying the equivalence class to which it belongs for each of the equivalence relations (provided the set of relations is sufficiently rich). Loosely speaking, we identify genes with a set of 'basic' equivalence relations and alleles with the corresponding equivalence classes. For instance, in a search space of faces, "same hair colour" and "same eye colour" might be two basic equivalence relations, which would induce the formae "red hair", "brown hair", "blue eyes", etc. Higher order formae are then constructed by intersection, e.g. "brown hair and blue eyes". Chromosomes made up of strings of alleles can then be used to *represent* the original structures of the search space (faces in our example).

These chromosomes make up the representation space and the objects they encode define the growth function.

For certain simple representations, the genes are orthogonal, meaning that any combination of allele values represents a valid solution and that implementation of appropriate genetic operators is straightforward, but in many cases this is not so (i.e. certain alleles are mutually incompatible). In other cases it is not easy to define equivalence relations (genes), so we simply identify particular subsets of the search space that share some characteristic, leading to chromosomes that consist simply of a set of "alleles". (For instance, in the TSP, we could identify $n(n-1)/2$ subsets of the search space, each containing all tours in which city i is linked to city j. Then a particular tour would be represented by the set of (undirected) edges it contained.) In still other problems, chromosomes may have variable length (Radcliffe, 1992a). It is primarily in these more complex situations that the benefits of formal analysis of representations becomes clear. We explore a more comprehensive taxonomy of formal representations in section 4.3.

4.1.1 Formal and Physical Representations

This thesis concentrates on representational aspects of search, and in doing so establishes some rather powerful, general results. Although we shall ignore the almost philosophical debate about whether it is *possible* to perform a search directly in the search space without a representation (which we can sidestep using an identity mapping between the space of chromosomes and the search space), it is important to distinguish between what we term formal and physical representations. A formal representation is introduced in order that move operators can be formulated (defined!) to act in the representation space $\mathcal{C}$, rather than the search space $\mathcal{S}$, and that some benefit is potentially conferred by this. The physical representation, on the other hand, is introduced in order to facilitate storage (or coding) of solutions on a computer or any other device, and is regarded as a completely separate issue.

Consider the concrete example of a three-dimensional Euclidean space $\mathcal{S}$ and the choice of representations between Cartesian coordinates (x, y, z) and spherical polars (r, θ, ϕ). We would consider the representation used by an algorithm to be defined *not* by the coordinate system in which the points in $\mathcal{S}$ were stored in a computer (whether in terms of high-level language constructs or actual discretised bit-patterns in memory chips), but rather by whether the move operators manipulate polar or Cartesian components. There are clearly problems for which each representation has advantages (e.g. the polar representation might be expected to offer benefits in spherically symmetric problems).

A failure to distinguish between formal and physical representations has also led to a

great deal of confusion in the literature. For example, many genetic algorithms defined for functions of real-parameters treat the physical representation of the solutions as the genetic structure to be manipulated by the genetic operators, and surprise is then expressed when the resulting algorithms are ineffective (e.g. see Goldberg, 1990). Simply because we write a potential solution to a three-parameter problem as $(-1.1, 3, 6)$ does not mean that -1.1, 3 and 6 are (directly) the genetic material to be manipulated. Equally, simply because we happen to store these three values on a binary computer using an IEEE floating-point representation does not mean that it is those 'bits' which are the interesting genetic material. Instead, we seek to define a formal representation that captures the problem structure of interest in order to generate genetic move operators which result in effective search strategies. Chapter 8 presents a more detailed exposition of real-parameter representations, clearly highlighting the somewhat bizarre nature of genetic operators and algorithms derived from physical rather than formal representations.

4.2 Generating Representations by Characterising Problem Domains

As outlined in section 4.1, we propose to generate formal representations based on equivalence relations defined over members of the search space. Figure 4.1 illustrates how this is effected through a problem-dependent *characterisation*, χ, that generates the required equivalences for any instance of a given problem. The characterisation is a mathematical device used to state our beliefs about problem structure; for example by identifying features of solutions thought to be related to performance. As will be shown in chapter 6, this characterisation captures all of the structure that will be exploited by a search algorithm. That same chapter also suggests quantitative methods for choosing between alternative characterisations for a particular problem.

Although the selection of an appropriate characterisation for a particular problem domain is an open problem, several *design principles* have been previously proposed (Radcliffe, 1991a). The most important of these is that the generated formae should group together solutions of related fitness (Radcliffe & Surry, 1994b), in order to create structure which can be exploited by the move operators. (It must also be possible to find a member of any given formae in "reasonable" time without resorting to enumeration, but this is true of most "reasonable" characterisations.) These ideas are explored in section 4.2.1 and then the techniques are illustrated for several simple problems. More extensive case studies for particular problem domains are presented in chapters 7 and 8.

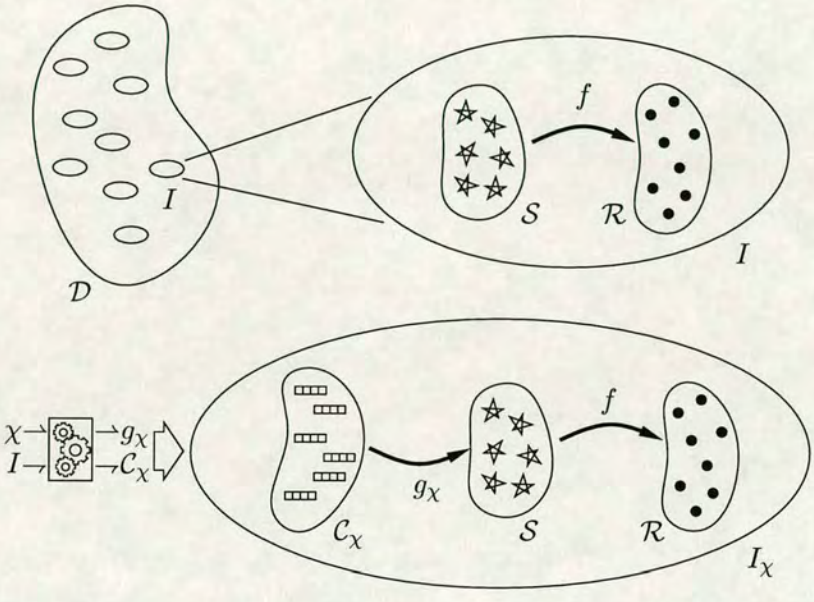


Figure 4.1: A *problem domain* $\mathcal{D}$ consists of a set of *problem instances*. Each instance I defines a *search space* $\mathcal{S}$ of candidate solutions, a *fitness function* f and a set of objective values $\mathcal{R}$. A *characterisation* χ of the domain specifies a set of equivalences among the solutions for any instance I. These equivalences induce a *representation* made up of a *representation space* $\mathcal{C}_\chi$ (of *chromosomes*) and a *growth function* g_χ mapping chromosomes to the objects in $\mathcal{S}$. A chromosome x is a string of *alleles*, each of which indicates that x satisfies a particular equivalence on $\mathcal{S}$. Algorithms can be completely specified by their action on the alleles of these generalised chromosomes, making them totally independent of the problem domain itself.

4.2.1 Forma Analysis

We have proposed generating formal representations by characterising particular features of solutions within a given problem domain that are thought to be relevant to fitness. Such a representation can be viewed as a mechanism for uniquely identifying any individual solution through listing the features of interest that it possesses. Arbitrary features can be defined by specifying the subset of solutions that have that feature. In the work of Holland (1975), subsets of a search space of k-ary strings (typically limited to $k = 2$) were identified using *schemata*—sets of strings sharing particular allele values. Radcliffe (1991a, 1991b) then generalised this idea to arbitrary search spaces with arbitrary commonalities using forma analysis. In Radcliffe's work, subsets of the search space, $\mathcal{S}$, thought to contain solutions with related performance are identified as *formae*; possibly as partitions[1] generated by equivalence relations, or simply as groups of solutions sharing some characteristic.

A characterisation, χ, of a problem domain is then simply a recipe for defining such a set of formae for any problem instance from some problem domain of interest. In many cases, χ takes the form of a set of equivalence relations Ψ for any problem instance, with formae defined as the equivalence classes induced by Ψ. (An equivalence relation $\psi : \mathcal{S} \times \mathcal{S} \longrightarrow \{0, 1\}$ can be thought of as a function which returns 1 if a given pair of solutions are "equivalent" and 0 otherwise.) Any solution can then be identified by specifying the equivalence class to which it belongs for each of the equivalence relations. Loosely speaking, we identify genes with a set of basic equivalence relations (from which any member of Ψ can be constructed) and alleles with the corresponding equivalence classes (see Radcliffe, 1994 for details).

More generally, a forma is simply any collection of solutions in which the corresponding chromosomes share particular alleles. Formae are typically given names such as ξ, and specified through a *description set* denoted $\langle \xi \rangle$. The description set is simply the set of alleles that a chromosome must have in order for the solution it represents to be a member of the forma in question, and is thus closely related to the set of defining positions (and defining values) familiar from schema analysis. Since the formal chromosome used here is precisely the set of alleles, it is clear that

$$\xi = \{a \in \mathcal{S} \mid \rho(a) \supset \langle \xi \rangle\}. \tag{4.1}$$

The selection of an appropriate characterisation for a particular class of problem must be driven by external knowledge of the structure of that problem class. However, once any suitable characterisation χ is given, we are able to use a formulaic procedure to generate the required equivalence classes for any given problem instance. This in turn allows us to derive problem-specific operators that reflect our beliefs about problem

[1] A partitioning of a set is a collection of disjoint subsets (partitions) covering the set.

structure through the way in which they manipulate solutions. For example, in the travelling sales-rep problem, we might reasonably believe that tours sharing any given edge will have related performance (indeed this is clear since the length of a tour is simply the sum of its edge lengths). Thus we might propose a characterisation which generates formae based on equivalence relations of the form "has edge $\overline{xy}$" (although other characterisations are also possible; see Radcliffe & Surry, 1994b).

Although we cannot give a general definition of an appropriate problem characterisation, we can formulate several desirable features of such characterisations (Radcliffe, 1991a). By constructing representations in which formae partition the search space in consistent ways, we are able to define genetic operators independent of any specific representation that will manipulate solutions in effective ways. These ideas are broadly similar in intent to the principle of meaningful building blocks suggested by Goldberg (1989c):

> The user should select a [representation] so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other [defining] positions.

This principle seeks to encapsulate some of the conditions that might be thought to help recombination to operate effectively. In more detail, recommended guiding principles include:

- *Correlation within formae.* The formae should group together solutions of related fitness. One measure of this is the mean fitness variance over formae of a given size. This has been measured experimentally for four representations in the travelling sales-rep problem and shown to correlate well with performance (Radcliffe & Surry, 1994b).

- *Minimal degeneracy*[2]. The number of distinct chromosomes representing each member of $\mathcal{S}$ should be small.

- *Computability.* It must be computationally efficient to exhibit a random solution contained within any specified forma. This is particularly important for non-orthogonal representations (see section 4.3) in which efficient patching operators are required in order to construct recombination and mutation operators (see chapter 5), but it applies to orthogonal representations as well (where the requirement is essentially that the growth function be computationally efficient). For example, in the TSP one could choose χ that defines equivalences between

[2]Other authors substitute the term *redundancy*, but this is here reserved for the situation in which chromosomes contain more information than strictly necessary to specify the solution they represent; see section 4.3.3.

all tours of length less than any constant c, but (unless P=NP) it is computationally infeasible to exhibit a tour from any particular formae. (Indeed, given a polynomial-time algorithm which simply indicates whether there is any tour with length less than any constant c, it is straightforward to construct a polynomial-time algorithm to solve the TSP). These restrictions have not been investigated in depth, but have not been a problem with those representations investigated to date.

4.2.2 Illustration: Grouping and Scheduling Problems

Consider a generic grouping problem, in which we seek an optimal partitioning of a universal set of n elements. In particular, consider the case in which the labelling and ordering of the partitions does not affect the fitness of a candidate partition. Practical examples of this type of problem would include equal-size bin-packing, symmetric process scheduling and so forth.

As pointed out by Falkenauer (1994), the "obvious" encodings that might be chosen to employ evolutionary algorithms with such problems lead to numerous different chromosomes encoding solutions with identical fitness. (He calls this redundancy, but for reasons explained in section 4.3 the alternative term degeneracy is preferred.) For example, consider a universal set of $n = 5$ elements. We can form at most five distinct groups, so might use a representation with five genes each of which could take one of five values (say the letters A through E). That is, the partition

$$\{1|2,3,4|5\} \tag{4.2}$$

might be represented as the chromosome

$$(A,B,B,B,C). \tag{4.3}$$

Note however that it could equally well be represented by (E,A,A,A,B) which shares no genetic material whatever with the first chromosome! Alternative representations proposed for this problem suffer similar problems; for example, systems based on decoding a permutation.

Using the techniques outlined in this chapter, it is straightforward to construct systematically a more appropriate representation that avoids these difficulties. We begin by mathematically characterising our beliefs relating problem structure to fitness; namely, that it is only the number and composition of the partitioned groups which affect fitness. One way of formalising this is to note whether any specific pair of elements are grouped together in a candidate solution. That is, given two elements i,j of the universal set, we say that two solutions a and b are equivalent if i and j are found in the same group in both a and b. Formally, for a problem instance I with universal set

of size n, $\chi(I)$ generates the $n(n-1)/2$ equivalence classes

$$\xi_{ij} = \{a \in \mathcal{S} \mid i, j \text{ grouped together in } a\}. \tag{4.4}$$

Any solution a would then be described by a set of alleles indicating to which of these subsets it belonged. For instance, with $n = 5$, the partition

$$a = \{1|2,3,4|5\} \tag{4.5}$$

would be described by the chromosome

$$x = \{23, 34, 24\}, \tag{4.6}$$

indicating that $a \in \xi_{23}$, $a \in \xi_{34}$ and $a \in \xi_{24}$. Note that this uniquely identifies the chromosome x with the solution $a \in \mathcal{S}$.

We could extend our characterisation to support the definition of genes by introducing an additional set of equivalences that note which pairs of elements are *not* grouped together in candidate solutions. Here, given two elements i, j, two solutions a and b are also equivalent if i and j are not in the same group in both a and b. Thus $\chi(I)$ generates equivalence relations of the form

$$\psi_{ij}(a,b) = \begin{cases} 1, & \text{if } i,j \text{ grouped together in } a,b \\ & \text{or } i,j \text{ not together in } a,b \\ 0, & \text{otherwise.} \end{cases} \tag{4.7}$$

These induce $n(n-1)$ equivalence classes, two corresponding to each equivalence relation, as follows:

$$\begin{aligned} \xi^1_{ij} &= \{a \in \mathcal{S} \mid i, j \text{ grouped together in } a\}, \\ \xi^0_{ij} &= \{a \in \mathcal{S} \mid i, j \text{ not together in } a\}. \end{aligned} \tag{4.8}$$

The partition

$$a = \{1|2,3,4|5\} \tag{4.9}$$

would now be described by $n(n-1)/2$ genes corresponding to the equivalence relations, each with a binary allele, yielding the chromosome

$$x = \begin{bmatrix} - & 0 & 0 & 0 & 0 \\ - & - & 1 & 1 & 0 \\ - & - & - & 1 & 0 \\ - & - & - & - & 0 \\ - & - & - & - & - \end{bmatrix}, \tag{4.10}$$

where the (i,j)th element (gene) of the array indicates to which equivalence class (allele) a belongs with respect to ψ_{ij}.

Given either of these characterisations, and the resulting representations (or other alternatives!), it is a simple matter to instantiate the template genetic operators described in chapter 5 and produce evolutionary algorithms that manipulate solutions

using exactly the (explicit) suppositions which were mathematically formalised to create the representation. The advantages of such an approach over either designing problem-specific genetic-like operators or inventing a growth function mapping from binary strings to elements of $\mathcal{S}$ seem clear.

In passing, note that scheduling (sequencing) problems can be dealt with using techniques similar to the grouping problems discussed above. Here the goal is to find an optimal ordering of n elements. Although these can be treated like ordinary permutations (for instance using operators like those defined for the TSP in chapter 7), it is not clear that these are the most appropriate. For example, Bierwirth *et al.* (1996) develop specialised scheduling operators such as PPX, which attempt to preserve precedence relationships, in preference to standard permutation operators. Again, it is straightforward to arrive at similar operators using the techniques outlined herein. If, for instance, we have candidate permutations π and π' and $\mathrm{idx}(\pi, i)$ gives the index of the ith element in π, we could choose a characterisation χ that explicitly captures precedence relationships by generating equivalence relations of the form

$$\psi_{ij}(\pi, \pi') = \begin{cases} 1, & \text{if } \mathrm{idx}(\pi, i) < \mathrm{idx}(\pi, j) \text{ and } \mathrm{idx}(\pi', i) < \mathrm{idx}(\pi', j) \\ & \text{or } \mathrm{idx}(\pi, i) > \mathrm{idx}(\pi, j) \text{ and } \mathrm{idx}(\pi', i) > \mathrm{idx}(\pi', j) \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

This gives rise to a representation qualitatively similar to that of the grouping representation studied above. For instance, with $n = 4$ the permutation

$$(2, 3, 1, 4) \tag{4.12}$$

would be represented as

$$x = \begin{bmatrix} - & 0 & 0 & 1 \\ - & - & 1 & 1 \\ - & - & - & 1 \\ - & - & - & - \end{bmatrix} \equiv \{1 \succ 2, 1 \succ 3, 1 \prec 4, 2 \prec 3, 2 \prec 4, 3 \prec 4\} \tag{4.13}$$

where the (i, j)th element (gene) of the array indicates to which equivalence class π belongs with respect to ψ_{ij}. (This is in fact the *precedence matrix* defined in Fox & McMahon, 1991 and used to treat the TSP.) Instantiating the representation-independent recombination operators described in chapter 5 results in operators almost identical to PPX.

4.2.3 Other Examples

Forma analysis has been used to characterise and construct representations for numerous problem domains, including subset-selection problems, scheduling problems, set partitioning, the travelling sales-rep problem, and real-parameter optimisation. Extensive case studies are presented for the latter two examples in chapters 7, and 8

Representation	Basic formae	Genes	Degeneracy		
			Orthogonal	Redundancy	
Binary-coded reals	value has ith bit equal to j	yes	yes	none	none
Dedekind real parameters	value above/below cut at i	yes	no	none	huge
TSP: City positions	city i in position j	yes	no	$\times 2n$	low
TSP: Undirected edges	contains link ij	no	no	none	low
Subset-selection: inclusive	includes ith element	no	yes	none	none
Subset-selection: incl/excl	incl/excl ith element	yes	yes	none	none

Table 4.1: This table summarises the characteristics of several representations for different problem domains. 'Basic formae' indicates the way in which basic subsets of the search space are identified, and the existence of genes is noted. Orthogonal representations are those in which any combination of alleles defines a valid solution. Degeneracy occurs when multiple chromosomes represent the same solution, and redundancy is the amount of excess information in the chromosome.

respectively. Table 4.1 contrasts several of these representations, including the traditional binary representation for real parameters, the "Dedekind" representation for real parameters introduced in Surry & Radcliffe (1996c), two natural representations for the travelling sales-rep problem, and two representations for subset-selection problems (used in neural-network topology optimisation), one in which only set membership is considered to be important and one in which both membership and non-membership is used (Radcliffe, 1992a). The table should be interpreted in conjunction with section 4.3 which defines the terms used to classify the various representations.

4.3 Taxonomy of Representations

Due to the generality of forma analysis, we are able to construct a wide variety of different types of representations. In this section we explore some of the ways in which such representations can be characterised.

4.3.1 Orthogonality

Informally, a representation is termed orthogonal if the value of each gene can be set independently of the values of all the others. In this situation, every potential chromosome corresponds to a legal solution. Although much of the literature on evolutionary algorithms focuses on orthogonal representations (such as the binary strings favoured by Holland, 1975; Goldberg, 1989c), many real-world optimisation problems are best attacked using non-orthogonal representations.

The traditional approach in this case is to use an orthogonal completion of the representation, along with standard genetic operators that may produce illegal genomes. Either a repair mechanism is then used to convert illegal chromosomes to legal ones, or

a penalty function is used to make illegal chromosomes appear "unfit" (effectively turning an unconstrained problem into a constrained orthogonal one). However, one of the attractions of using using the formal techniques described herein is that it becomes possible to derive problem-specific genetic operators that will not generate illegal genomes for these representations, allowing (we contend) more efficient optimisation directly within the search space.

As a concrete example, take the travelling sales-rep problem described in more detail in chapter 7. All of the "natural" representations for this problem are non-orthogonal, whether based on what edges are contained in a tour, or what permuted order the cities are visited in[3]. Although naïvely one might hope to extend the search space in order to use a simple orthogonal representation (for instance employing a penalty function to avoid convergence to an infeasible solution—see chapter 11), such an extended search-space typically contains only a tiny fraction of legal solutions. Consider for instance an orthogonal edge representation in which each solution is represented by a chromosome with a single binary gene for each of the possible $n(n-1)/2$ edges. Feasible tours correspond to chromosomes with exactly n 1's in this representation, with the further constraint that the edges form a closed tour of all the cities. In fact, we know that there are exactly $(n-1)!/2$ such chromosomes of the $2^{n(n-1)/2}$ possible chromosomes in this representation. Using Stirling's formula, $n! \simeq \sqrt{2\pi n}(n/e)^n$, and noting that $2^{n/2} \gg (n-1)/e$, it is clear that the proportion of feasible chromosomes is vanishingly small for practical n. It thus seems likely that any evolutionary algorithm based on such a representation will spend most of its time penalising inadmissable solutions, and very little finding good feasible ones.

4.3.2 Allelic and Genetic Representations

The notions of genes and alleles are very familiar, but need to be defined rather carefully for present purposes. A distinction will be drawn between *genetic* representations and *allelic* representations. The former typically arise when the problem domain is characterised by equivalence relations (as a solution will be a member of a specific equivalence class with respect to each equivalence relation), while the latter arise when the problem domain is characterised using a collection of equivalence classes that do not directly form complete partitionings.

A formal genetic representation is precisely a formal version of the familiar string composed of genes, and should cause little confusion. It will be assumed that a genetic representation consists of a string of n *genes,* numbered 1 to n, and that each gene

[3]We can certainly construct unnatural orthogonal representations, for example with the first allele indicating which of the n cities to visit first and the ith allele denoting which of the $n-i+1$ cities not already in the tour to visit next, but it is doubtful that such context sensitivity would aid useful search.

takes on values from some (typically but not necessarily finite) set $\mathcal{A}_i$. Thus in the case of a genetic representation, the representation space will be assumed to have the form

$$\mathcal{C} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n, \tag{4.14}$$

so that a chromosome is formally a vector of gene values. As before, it will not be assumed that all members of $\mathcal{C}$ correspond to solutions in the search space $\mathcal{S}$ (the representation may be non-orthogonal).

A formal *allele* in the context of a genetic representation will be considered to be an ordered pair consisting of a gene and one of its possible values, so that a chromosome $x = (x_1, x_2, \ldots, x_n)$ has alleles $(1, x_1), (2, x_2), \ldots, (n, x_n)$. This formulation of alleles, so far from being new, was suggested in Holland (1975), albeit with different motivation.

There are situations in which a suitable genetic representation of the form described above is not straightforwardly available. In such situations, it may be appropriate to drop the requirement that genes be defined, working instead with an *allelic representation* (introduced in Radcliffe & Surry, 1994c). In such a representation, instead of being a vector, a chromosome is a *set* whose elements are drawn from some universal set $\mathcal{A}$. One of the key features of allelic representations is that genes are not formally well-defined, so that ordinary gene-based notions of recombination are not applicable.

In order to qualify as a formal allelic representation, all that is necessary is that the growth function g of equation 3.2 be surjective, as required previously, and that $\mathcal{C} \subseteq \mathbb{P}(\mathcal{A})$, where $\mathbb{P}(\mathcal{A})$ denotes the power set (set of all subsets) of $\mathcal{A}$. Again, there is no requirement that all members of $\mathcal{C}$ represent solutions in $\mathcal{S}$. (As noted in section 3.1, $\mathcal{S}$ is assumed to be finite, with infinite search domains treated by discretisation as discussed in chapter 8.)

More concretely, consider characterisations of the TSP based on edges (city-to-city links), as it is clear that the edges present in a tour are the key (only!) determinant of its fitness. If such edges are considered to be directed, then a genetic representation is generated simply by letting the ith gene take the value of the city visited after city i, so that $(4, 3, 1, 2)$ represents the tour that goes from city 1 to city 4, to city 2, to city 3, and back to city 1. If, however, the edges are considered to be undirected (so that the 3–2 edge and the 2–3 edge are equivalent) it is no longer straightforward to identify genes, because each city is connected to two others. In this case, one approach is simply to let $\mathcal{A}$ be the set of all possible edges and represent a tour by the set of (undirected) edges it contains. The tour represented by $(4, 3, 1, 2)$ in the directed-edge representation is then represented by $\{14, 24, 23, 13\}$ in the undirected edge representation, where the edges have all been written with the lower-numbered city first to emphasize their directionless nature. Note that neither of these representations are orthogonal. The problem of choosing between these (and other) characterisations for the TSP is discussed at length in chapter 7.

It is obviously trivial to construct an allelic representation from a genetic representation by taking $\mathcal{A}$ to be the set of all alleles, so that (referring to equation 4.14)

$$\mathcal{A} = \bigcup_{i=1}^{n} \mathcal{A}_i. \tag{4.15}$$

Under this scheme a solution is represented simply by its set of (formal) alleles, so that $(4, 3, 1, 2)$ in the directed edge representation gives rise to $\{(1, 4), (2, 3), (3, 1), (4, 2)\}$ in the allelic representation. This motivates the term "allelic representation", and the members of $\mathcal{A}$ will henceforth be referred to as alleles whether they are alleles in the sense of ordered pairs of gene values from a genetic representation or simply members of a given set $\mathcal{A}$ used directly to construct an allelic representation.

It is only slightly less obvious that given an allelic representation it is also easy to construct a genetic representation from it by creating for each member of $\mathcal{A}$ a binary gene that takes the value 1 if the (allelic) chromosome contains that allele and 0 if it does not. It should be noted, however, that such an induced genetic representation is very different from the initial allelic representation, so much so that if an allelic representation is then constructed from the (induced) genetic representation it will be quite different, in general, from the original allelic representation. (This is clear because we have effectively introduced new information into the chromosome by indicating which of the original alleles were not present, in addition to noting those that were present.) For this reason, most of the operators introduced in chapter 5 are defined with respect to allelic representations, which allows them to be used for (natural) allelic representations or genetic representations without complication.

4.3.3 Redundancy and Degeneracy

In order to classify representations further, it is useful to introduce the distinct but oft-confused notions of redundancy and degeneracy.

A representation is said to contain *redundancy* when not all of the genetic information contained in a chromosome is strictly necessary in order to identify uniquely the solution to which it corresponds. For example, if a solution is a set n of numbers that must sum to some particular value v, and all n values are stored in the representation of a solution, there is redundancy because the value of any single member of the set can be deduced from the other members.

Formally, a representation is redundant if there exists a chromosome $x \in \mathcal{C}$ that can be uniquely identified with $g(x) \in \mathcal{S}$ by a subset of its alleles. That is, if $x = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is such a chromosome, then there is a subset of its alleles $\hat{x} \subset x$ with the property that $\forall y \in \mathcal{C}, \hat{x} \subset y \Longrightarrow g(y) = g(x)$.

There is a close relationship between redundancy in a genetic representation and the degree of independence of its genes, with non-independence often leading to redund-

ancy (though not always—consider the situation in which the ternary chromosomes $\{11, 12, 22, 23, 33, 31\}$ represent distinct legal solutions but $\{13, 21, 32\}$ are illegal). Similarly, redundancy typically occurs in representations that are not orthogonal, with high redundancy necessarily giving rise to a high degree of non-orthogonality, but it is not clear to what extent—if any—this inhibits the search. (However, redundancy and orthogonality are not mutually exclusive—consider a two-bit binary representation with growth function $g(00) = g(01) \neq g(10) \neq g(11) \neq g(00)$.)

In contrast, a representation is said to exhibit *degeneracy* if more than one chromosome can represent the same solution, (i.e. if the genotype-phenotype mapping is non-injective). Degeneracy is widely, but not universally, perceived as detrimental to genetic search (Radcliffe, 1993).

4.3.4 Linkage Considerations

The *linkage* of a collection of genes refers to its likelihood of being transmitted *en masse* from one parent to a child under the action of recombination. Under the action of N-point crossover, for instance, a group of genes is said to be *tightly linked* if they are close together on the genome, and thus relatively unlikely to be disrupted by crossover.

Holland (1975) originally intended that one-point crossover be used with locus-independent genetic representations. In this case the locus (position on the genome) of an allele does not define its meaning (i.e., the gene to which it corresponds). Instead, alleles carry the gene to which they correspond as a label, so that the allele (3,4) might be used to indicate that the third gene takes the value four, as above. Such a coding scheme allows alleles to be moved around the genome adaptively, with the aim of evolving linkages between alleles that reflect the degree to which they affect one another, strongly interacting alleles moving close together over time to reduce their likelihood of disruption. Holland proposed the inversion operator to achieve this adaptive re-linking.

Note that this definition of linkage assumes both that genes exist (in order that corresponding alleles can be identified and aligned between parents during crossover), and that the genes are ordered linearly within the chromosome. Both of these requirements can be ameliorated for the generalised representations considered here. For instance, with allelic representations, in which genes are not defined, we could impose an order (either fixed or evolving) in which to consider alleles from each parent rather than the normal randomised approach (e.g. see the definition of RAR in chapter 5). Further, there seems no reason in general that the ordering of genes (or alleles) need be linear—it is perfectly reasonable to consider utilising either a tree-structure or more general neighbourhood structure. A particularly interesting area future investigation is the application of our formal methodology to genetic programming, in which linkage considerations will feature heavily.

Although locus-independent representations and operators are now used only rarely in genetic algorithms, the notion of linkage is important, and many believe that as problem complexities and run lengths increase these ideas may be revived (e.g. Harik & Goldberg, 1996).

4.4 Discussion

In this chapter, we have presented a formulaic methodology for generating representations for arbitrary problem domains given a characterisation of that problem domain. This characterisation explicitly encapsulates our beliefs about problem structure, and ensures that an evolutionary algorithm based on such a representation acts in accordance with these beliefs. Constructing an effective characterisation is primarily guided by identifying features of solutions related to performance, but is necessarily a problem-dependent task for which no generic recipe can be provided. However, we have demonstrated the approach for several realistic problem domains, and discussed a partial taxonomy of the types of representation that arise in practise. Further insight into choosing between alternative characterisations for a fixed problem domain will be presented in chapter 6.

It should be pointed out that representation is often given an even more significant rôle than the central one discussed above, by exploiting the developmental (growth) process (Hart *et al.*, 1994). Davis (1991b), for example, has long used devices such as "greedy decoders" to map a "chromosome" manipulated by a genetic algorithm to the solution that it represents. While this can formally be viewed as a radically different kind of search, in which the search objects are really "hints" or starting points for a full local search, the more intuitive picture of a particularly complex growth function has some merit. Of course, fascinating new issues arise in this context, as there is typically no particular reason to suppose that the decoder is surjective (indeed, it would almost defeat its motivation if it were!), nor even that it can generate points of interest (such as global optima) for *any* input. Moreover, the decoder may well contain a stochastic element, further complicating matters. It nonetheless remains to generalise our theories of search to begin to accommodate some of these less conventional, but manifestly powerful, complex representations.

Chapter 5

Formal Algorithms

It sounded an excellent plan, no doubt, and very neatly and simply arranged: the only difficulty was, that she had not the smallest idea how to set about it; ...
(Lewis Carroll)

Traditional evolutionary algorithms are typically defined using a set of move operators which assume a particular form of the representation space. For example, many genetic algorithms assume chromosomes are binary strings, and most evolution strategies assume chromosomes are strings of real parameter values. Although some of the operators used by such algorithms can be generalised straightforwardly to related representation spaces (for example, the generalisation of N-point crossover to mixed arity chromosomes in Holland, 1975), they typically are not general enough to handle arbitrary representations. In particular, variable-length genomes and non-orthogonal representations both present difficulties, and have generally led in the past to *ad hoc* construction of problem-specific move operators (for example in the travelling sales-rep problem). This in turns means that the resulting evolutionary algorithms are problem-specific, making it difficult to transfer learnings between domains.

We aim in this chapter to define formal (evolutionary) algorithms that are completely independent of problem domain. It is clear that the primary difficulty in doing this is the specification of problem-independent move operators (as it is straightforward, for instance, to define a problem-independent replacement or selection methodology). However, by using the methodology of chapter 5, we are able to design formal, *representation-independent* move operators whose actions are specified by defining how they manipulate the formae-membership properties of chromosomes. In effect the resulting operators are thus parameterised by representation, and as such can be realised for *any* problem domain by instantiating them with a representation appropriate to that domain.

In section 5.1 we review a number of guiding principles for the design of representation-independent genetic operators based on the structure we attempt to capture when characterising a problem domain using forma analysis. These have led to the

development of several generalised recombination and mutation operators (sections 5.2 and 5.3).

The result of creating representation-independent forms for all commonly used classes of genetic operators is that we are able to construct formal representation-independent algorithms. For example:

A representation-independent evolutionary algorithm

1. Generate an initial population by randomly sampling p times from the space of chromosomes.
2. Evaluate the p members of the initial population via the growth and fitness functions.
3. Select two parents using (deterministic) binary-tournament selection.
4. Recombine the parents using RAR_2 (see section 5.2.3).
5. Mutate the resulting child using $\mathrm{BMM}_{0.1}$ (see section 5.3.1).
6. If the child does not exist in the population, evaluate it and replace the member with the worst fitness.
7. Repeat from step 3 until termination criterion.

Note that every step of the algorithm is precisely defined, and that given a representation of a problem domain, we can mathematically derive a concrete search strategy suitable for implementation on a digital computer (see chapter 6). This is different from traditional evolutionary algorithms, in which steps 4 and 5 would have to be modified for any problem domain which required a new representation space.

The primary advantage of constructing such algorithms is that it permits variation of representation as an independent variable in the context of a fixed formal algorithm, allowing the influence of representation to be isolated and measured. This is achieved by comparing the performance of algorithms that are identical in all respects other than the representation chosen—ones that execute the same reproductive plan, with the same set of operators and the same parameters. Chapter 6 pursues this idea, leading to the compelling practical examples developed in chapters 7 and 8.

5.1 Design Principles for Genetic Operators

Having selected a characterisation χ thought to generate formae grouping together solutions of related performance, we are able to motivate several *design principles* for constructing genetic operators (Radcliffe, 1991a; 1994). As outlined below, principles such as *respect, transmission, assortment,* and *ergodicity* suggest how the genetic "information" stored in parent chromosomes might best be "passed on" to child chromosomes in order to exploit our assumptions about performance within formae. (Although it is

certainly the case that not all of these principles are necessarily desirable for particular problems—indeed they are often mathematically incompatible.) Genetic operators based on these principles can then be precisely defined independently of any particular representation, by specifying how they manipulate the formae-membership properties of their operand(s).

Formally, any search operator, ω, can be viewed as generating a child chromosome from one or more parent chromosomes along with a control parameter selected uniformly from a control set (which provides "randomness"; see Radcliffe, 1994), thus

$$\omega : \mathcal{C}^q \times \mathcal{K}_\omega \longrightarrow \mathcal{C}, \tag{5.1}$$

where $\mathcal{K}_\omega$ is the set of possible control parameters (e.g. cross points, mutation masks, etc.), and q is the arity of the operator (so $q = 1$ for a mutation operator, $q = 2$ for a typical recombination operator, etc.). We can define an equivalent probabilistic form of ω, which samples uniformly from the control set:

$$\tilde{\omega}(x_1, \ldots, x_q) \triangleq \omega(x_1, \ldots, x_q, \kappa) \ : \ \kappa \sim U(\mathcal{K}_\omega). \tag{5.2}$$

For any given representation, we can use the definition of the operators to derive formally a problem-specific version of that operator. This leads in some cases to previously known operators in a given search domain, but in other cases leads to new insights about what form of recombination or mutation might be applicable to a given problem domain. The generalised operators are particularly relevant to non-orthogonal representations (in which not all combinations of alleles are valid), which often arise when the problem characterisation is based on beliefs which are not completely compatible.

The proposed design principles include:

5.1.1 Respect

Respect requires that children produced by a recombination operator $\mathcal{X}$ are members of all formae to which both their parents belong. For example, if there were equivalence relations about hair colour and eye colour in the set of equivalence relations, Ψ, defining the representation, then if both parents had red hair and green eyes, so should all children produced by $\mathcal{X}$.

More formally, a recombination operator $\mathcal{X} : \mathcal{S} \times \mathcal{S} \times \mathcal{K}_\mathcal{X} \longrightarrow \mathcal{S}$ (where $\mathcal{K}_\mathcal{X}$ is a set of control parameters such as cross-points or crossover masks) is said to *respect* the set of formae Ξ generated by Ψ iff

$$\forall \xi \in \Xi \, \forall a \in \xi \, \forall b \in \xi \, \forall \kappa \in \mathcal{K}_\mathcal{X} : \ \mathcal{X}(a, b, \kappa) \in \xi. \tag{5.3}$$

5.1.2 Transmission

A recombination operator is said to be *transmitting* if and only if each allele in every child it produces is present in at least one of its parents. In the case of a genetic

representation, then every child such an operator produces must be equivalent to one of its parents under each of the basic equivalence relations, and the operator is said to *transmit genes* (loosely, every gene is set to an allele which is taken from one or other parent). For example, if one parent had red hair and the other had brown hair, then transmission would require that the child had either red or brown hair.

It is easy to see that a recombination operator that transmits genes is respectful. However, with an allelic representation this is not necessarily the case. For example, the Edge Recombination Operator in its original form (Whitley *et al.*, 1989) sought to transmit as many undirected edges from the two parents as possible, and was thus striving to achieve allele transmission, which it did typically with over 99% success. The operator was then modified to achieve strict respect by placing all edges common to the parents in the child at the start (Whitley *et al.*, 1991b), resulting in a quite different (and apparently more successful) operator.

Formally, we require that such a recombination operator $\mathcal{X} : \mathcal{S} \times \mathcal{S} \times \mathcal{K}_{\mathcal{X}} \longrightarrow \mathcal{S}$ satisfies

$$\forall a, b \in \mathcal{S}\ \forall \zeta \in \Xi\ \forall \kappa \in \mathcal{K}_{\mathcal{X}} :\ \mathcal{X}(a, b, \kappa) \in \zeta \Longrightarrow a \in \zeta \text{ or } b \in \zeta \tag{5.4}$$

where the ζ are restricted to basic formae (that is $\nexists A \subseteq \Xi \setminus \{\zeta\} :\ \bigcap\{\xi \in A\} = \zeta$).

5.1.3 Assortment

Assortment requires that a recombination operator be capable of generating a child with any compatible characteristics taken from the two parents. In our example above, if one parent had green eyes and the other had red hair, and if those two characteristics are compatible, assortment would require that we could generate a child with green eyes and red hair.

Formally, a recombination operator is said to *properly assort* the formae generated by Ψ iff

$$\forall \xi_1, \xi_2 \in \Xi\ (\xi_1 \cap \xi_2 \neq \varnothing)\ \forall a_1 \in \xi_1\ \forall a_2 \in \xi_2\ \exists \kappa \in \mathcal{K}_{\mathcal{X}} :\ \mathcal{X}(a_1, a_2, \kappa) \in \xi_1 \cap \xi_2. \tag{5.5}$$

Note that in general it is not necessary for the operator to be able to generate a child with any combination of compatible characteristics in a single recombination: repeated incestuous recombination may be required. In the latter case, the operator is said to be *weakly,* rather than *properly,* assorting.

5.1.4 Ergodicity

The ergodicity principle demands that we select operators such that it is possible to move from any location in the search space to any other by their (finitely) repeated action. Typically a standard mutation $\mathcal{M}$ operator is sufficient, provided that it satisfies

$$\forall a, b \in \mathcal{S}\ \exists \kappa_1, \kappa_2, \ldots, \kappa_k \in \mathcal{K}_{\mathcal{M}} :\ \mathcal{M}(\cdots \mathcal{M}(\mathcal{M}(a, \kappa_1), \kappa_2), \ldots, \kappa_k) = b \tag{5.6}$$

5.2 Generalised Recombination Operators

Evolutionary computing has given rise to a large and growing collection of recombination operators. When the representations used are orthogonal, certain properties tend to be common to almost all such operators, and seem barely worthy of comment. When non-orthogonal representations are used, however, these properties are much less universal and become more salient. The three properties of respect, transmission and assortment described in section 5.1 are of particular relevance, and as we see below it is possible to employ them in order to define generalised, representation-independent recombination operators.

The most familiar crossover operators from genetic algorithms (such as N-point crossover; reduced-surrogate crossover, Booker, 1987; parameterised uniform crossover, Spears & De Jong, 1991; shuffle crossover, Schaffer *et al.*, 1989 etc.) are respectful, transmitting and assorting with respect to familiar string representations, which are typically orthogonal. However, for non-orthogonal representations assortment is often incompatible with respect and gene transmission. Many operators for non-orthogonal representations fail to have some or any of these properties.

Several representation-independent recombination operators have previously been introduced (Radcliffe, 1994), under the names of *random respectful recombination* (R^3), *random transmitting recombination* (RTR) and *random assorting recombination* (RAR). These are described in detail below—as the names suggest, R^3 is always respectful, RTR is always transmitting and RAR is always assorting, in each case with respect to the representation for which they are instantiated. For orthogonal representations, RAR and RTR are equivalent and thus assort, transmit and respect, and are in fact equivalent to uniform crossover. In the special case of binary representations, RTR reduces to R^3.

A *generalised N-point crossover,* GNX, has also been proposed (Radcliffe & Surry, 1994b; 1994c). This begins in much the same way as standard N-point crossover, dividing the two parents with N cut-points, and then using genetic material from alternating segments. The alleles within each segment are tested in a random order for inclusion in the child, and any remaining gaps are patched by randomly selecting compatible alleles first from the unused alleles in the parents, and then from all possible alleles. The need for patching of an incompletely specified child is a general feature of representation-independent recombination operators, and is discussed in more detail in section 5.2.5.

5.2.1 Random Respectful Recombination (R^3)

Random respectful recombination (R^3) is defined as that operator which selects a child uniformly at random from the set of all solutions which share all characteristics pos-

sessed by both parents (their *similarity set*). Formally

$$\forall a, b \in \mathcal{S} : \mathrm{R}^3(a, b) \sim U(\cap\{\xi \in \Xi \mid a, b \in \xi\}), \tag{5.7}$$

where $U(A)$ is the uniform distribution over the set A.

5.2.2 Random Transmitting Recombination (RTR)

The *random transmitting recombination* (RTR) operator is defined as that operator which selects a child uniformly at random from the set of all solutions belonging only to basic formae present in either of the parents (their *dynastic potential;* Radcliffe, 1994). Formally

$$\forall a, b \in \mathcal{S} : \mathrm{RTR}(a, b) \sim U(\{c \in \mathcal{S} \mid \forall \zeta \in \Xi, c \in \zeta \Longrightarrow a \in \zeta \text{ or } b \in \zeta\}) \tag{5.8}$$

where again ζ are restricted to basic formae.

5.2.3 Random Assorting Recombination (RAR)

The *random assorting recombination operator* (RAR_w), is a generalised form of uniform crossover. Informally, it proceeds to choose alleles from those of the parents, inserting them in the child when it can, and discarding them otherwise. If the parents' alleles become exhausted before the child is fully specified, its remaining alleles are set either at random (from among the legal combinations) or by some form of patching, as described in section 5.2.5.

As with uniform crossover, locus has no effect on the likelihood that a group of alleles will be inherited, and—neglecting the fact that alleles from one parent are known to be compatible, whereas those from different parents may not be—the number of alleles taken from each parent is binomially distributed. Indeed, in the limit of orthogonal genetic representations (those in which all allele patterns are legal) RAR_w reduces to uniform crossover (with parameter one-half).

Formally, RAR_w takes a parameter w that specifies a relative weighting between alleles common to the parents and those that are present only in one[1]. $\mathrm{RAR}_w(X, Y)$ begins by assigning to each allele $\alpha \in X \cup Y$ a weight $W(\alpha)$ given by

$$W(\alpha) = \begin{cases} w, & \text{if } \alpha \in X \cap Y, \\ 1, & \text{otherwise.} \end{cases} \tag{5.9}$$

It then initialises an empty child $Z_0 = \varnothing$ and selects an allele α_0 from $\mathcal{A}_0 \triangleq X \cup Y$, with probability proportional to its weight. This allele is added to the proto-child to form Z_1. The following process is then repeated for steps indexed by i:

[1]The parameter w effectively balances the trade-off between exploitation (of useful information contained in both parents) and exploration (of potentially useful new combinations of alleles). In the limit of $w = \infty$, RAR reduces to R^3, while RAR_0 maximises assortment.

Repeat until $\mathcal{A}_i = \varnothing$:

1. Let $\mathcal{A}_i \triangleq \mathcal{A}_{i-1} \setminus \alpha_i$.
2. Choose a new allele α_i from $\mathcal{A}_i$ with probabilities proportional to the weights of the alleles in $\mathcal{A}_i$.
3. Let $Z_i \triangleq \begin{cases} Z_{i-1} \cup \{\alpha_i\}, & \text{if } \alpha_i \text{ is compatible with those in } Z_{i-1}, \\ Z_{i-1}, & \text{otherwise.} \end{cases}$
4. $i \leftarrow i + 1$.

In step 3 above, "compatible" means that there exists a solution in $\mathcal{S}$ whose representative in $\mathcal{C}$ has all the alleles in Z_{i-1} and also α_i.

At this stage it is possible that the child will be completely specified, but in general this will not be the case. If it is not, a patching algorithm must be used to complete the child. The most general way to achieve this is to select randomly (uniformly) from the chromosomes that include all the alleles in the proto-child constructed thus far. Section 5.2.5 introduces more sophisticated patching operators.

5.2.4 Generalised N-point Crossover (GNX)

In constructing a generalised form of N-point crossover, it is convenient to consider only genetic representations. The difficulty in applying conventional crossover operators is that not all combinations of gene values are legal. Let $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_N\}$ be a set of cross points, with $0 < \ell_1 < \ell_2 < \cdots < \ell_N < n$. This breaks a parent (genetic) chromosome x into $N + 1$ segments

$$(x_1, x_2, \ldots, x_{\ell_1 - 1}),\ (x_{\ell_1}, x_{\ell_1 + 1}, \ldots, x_{\ell_2 - 1}),\ \ldots,\ (x_{\ell_N}, x_{\ell_N + 1}, \ldots, x_n), \tag{5.10}$$

and breaks up the second parent y into corresponding segments.

The first phase of GNX's operation uses the same genetic material as ordinary N-point crossover, i.e., alternate segments from the two parents. It proceeds by picking a random order to visit the $N + 1$ segments (irrespective of the parents to which these segments are assigned). Within each segment, the alleles are "tested" in a random order. An allele is "tested" by seeing whether it can be placed in the child—i.e. whether it is compatible with those alleles that have already been placed in it. If compatible, the new allele is inserted, otherwise it is discarded. Because in general after this process has terminated the child will still be incomplete, a second phase then commences in which the genetic material discarded by ordinary N-point crossover (the 'complementary' alternating sections) is used to try to fill in any gaps. The segments are again visited in a random order and the alleles within them are tested in random sequence. If the child is still incomplete after this, the child is completed at random or by some other patching method. The general pattern of progress of GNX is shown in figure 5.1.

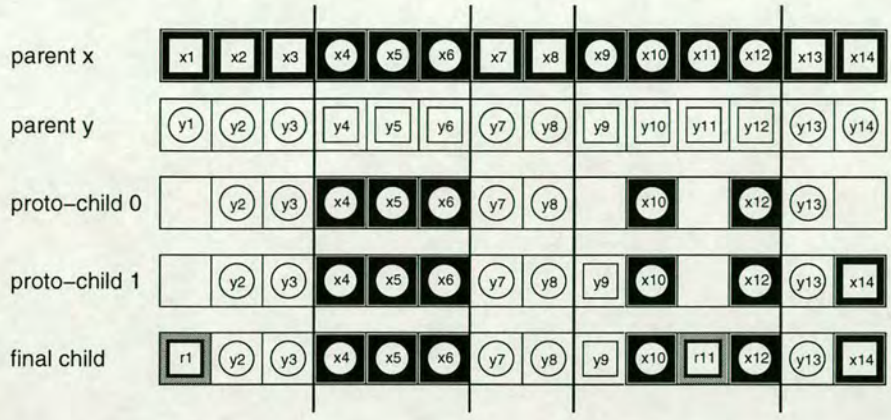


Figure 5.1: GNX first copies gene values from alternating segments (circled) of the parent chromosomes, visiting the segments and testing the genes within these segments in a random order. Gene values are copied to the child only if they are compatible with those already present. In this example, genes 2–8, 10, 12 and 13 are assigned in this way, resulting in proto-child 0. For genes not set by this process, alleles from the unused segments (boxed) of the parents are then tested for inclusion, again in random sequence. In the example, genes 9 and 14 are assigned thus, to give proto-child 1. Genes still not fixed after this process are assigned either at random from the set of legal combinations, or by some heuristic or other patching procedure. In this example, genes 1 and 11 fall into this category.

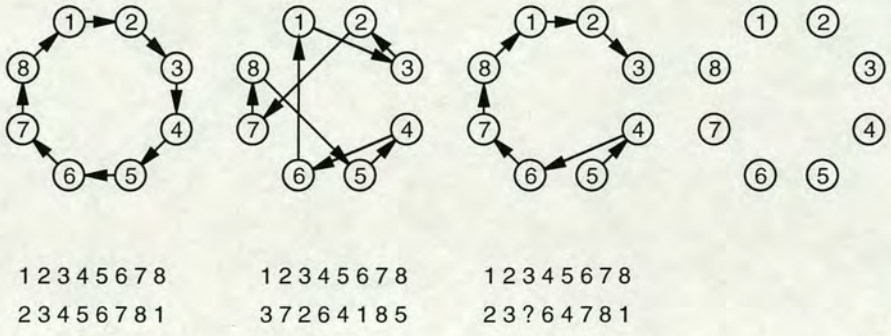


Figure 5.2: Two parent tours, one possible partial child they can produce under G2X, and an empty grid for the reader to use while following through the example.

Figure 5.2 illustrates the action of G2X with the directed edge representation for the travelling sales-rep problem (discussed in detail in chapter 7). In this representation, the ith gene represents a directed edge from city i to city x_i. The figure shows one potential child produced by the parents given by

$$\begin{array}{rcl} x & = & (2,3,4 \mid 5,6,7 \mid 8,1), \\ y & = & (3,7,2 \mid 6,4,1 \mid 8,5). \end{array} \tag{5.11}$$

with cross points chosen at 3 and 6.

Suppose the permutation of the segments chosen is $(3,2,1)$. Then the third segment of x (visited first) will be inserted whole, giving edges $7 \rightarrow 8$ and $8 \rightarrow 1$ (or the proto-child $(\square,\square,\square,\square,\square,\square,8,1)$). Then alleles in segment 2 from y will be tested in a random order, say $5 \rightarrow 4, 6 \rightarrow 1, 4 \rightarrow 6$ and the first and third (in this case) will be accepted, giving the proto-child $(\square,\square,\square,6,4,\square,8,1)$. The first segment of x is then tested, and the edges $1 \rightarrow 2$ and $2 \rightarrow 3$ will be accepted giving $(2,3,\square,6,4,\square,8,1)$. This completes the first phase.

The untested segments are then visited in random order, say first $(5,6,7)$ from x, then $(3,7,2)$ from y, and finally $(8,5)$ from y. During this process only the edge $6 \rightarrow 7$ will be accepted, giving the proto-child $(2,3,\square,6,4,7,8,1)$.

Since this child is still incomplete, it must be patched. In this case however, only one legal chromosome (with directed edges) has the required allele pattern, namely $(2,3,5,6,4,7,8,1)$, so it would be the result of the cross.

5.2.5 Patching by Forma Completion

For general representations, the recombination operators defined above produce only a partially-specified *proto-child* that must be completed in some manner to generate the final child. In the case of genetic representations, it is natural to think of the proto-child as a generalised schema to which the final child must belong. For allelic representations this may seem less natural, as the proto-child is specified only by a set of alleles it must contain. However, recalling that containing an allele is equivalent to belonging to the basic forma associated with that allele, we again see that the proto-child is simply a forma (constructed as an intersection of basic formae) from which the final child is chosen.

Formally, then, the first stage of a generalised recombination operator produces a proto-child $A \subseteq \Xi$ which is a collection of formae in which the final child a must be contained. Thus the final child a must lie within the subset of solutions defined by:

$$\mathcal{S}_a \triangleq \cap\{\xi \in A\}. \tag{5.12}$$

The simplest method of choosing a final child, called *random forma completion* is simply to select the child a uniformly at random from $\mathcal{S}_a$, i.e.

$$a \sim U(\mathcal{S}_a). \tag{5.13}$$

Alternative patching methods which are directed, in the sense that they attempt to choose a high-quality child from $\mathcal{S}_a$, can also be considered. One option is to choose the best solution consistent with the proto-child. In general, this would be prohibitively expensive, but if the number of unspecified alleles were small and the fitness function were decomposable this method can reasonably be considered. This method is termed *globally optimal forma completion.* Here,

$$a = \arg\max_{b \in \mathcal{S}_a} f(b). \tag{5.14}$$

A more practical method in many circumstances is to find a local optimum within the forma. With *locally optimal forma completion,* this is achieved by completing the forma at random and then testing minimal mutations (see section 5.3) that remain within it in sequence, accepting any that are better. This process continues until there is no minimal mutation within the forma that is better than the current solution.

5.3 Generalised Mutation and Hill-climbing Operators

Since mutation is widely recognised as playing a vital rôle in evolutionary search it is clearly essential to define representation-independent forms for such operators if we are to realise our goal of specifying representation-independent evolutionary algorithms. It is also clear that by defining representation-independent forms of mutation, the construction of generalised hill-climbers becomes straightforward. The present section attempts to set the discussion of representation-independent mutation operators in a reasonably general context, using a particular problem—the TSP—to illustrate the actions of the operators.

In a general representation, it will often not be possible to use standard gene-wise mutation operators because simply replacing one allele with a randomly chosen alternate may may well lead to an inconsistent chromosome. As with recombination then, we must define representation-independent forms of mutation quite carefully. A number of considerations affect the formulation of such an operator, including the characteristics of mutation operators in normal use, the perceived function of mutation and the behaviour desired of a generalised mutation operator in special limits (such as the case of orthogonal representations). The different strands of evolutionary computing use rather different sorts of mutation operators. One nearly universal characteristic, however, is that they ensure ergodicity, i.e. that the entire search space remains accessible from any population, and indeed from any individual. In most cases mutation operators can actually move from any point in the search space to any other point directly, but the probability of making "large" moves is very much smaller than that of making "small" moves (at least with small mutation rates). For example, in evolution strategies

mutation is typically Gaussian on a parameter-by-parameter basis, while in genetic algorithms with orthogonal representations a probability of (uniformly) choosing a new value for each gene is most commonly used, giving rise to a binomial distribution for the number of mutations made. It is a characteristic of most mutation schemes that relatively unlikely "large" moves can be effected by a series of smaller moves, thus making large, potentially fruitful moves possible with reasonable probability through iteration provided that the intermediate points (solutions) are themselves viable in the context of the reproductive plan used.

In order to clarify notions of "large" and "small" mutations, a metric (distance measure) will be introduced over the representation space $\mathcal{C}$, leading to the concept of a *minimal mutation*. In the case of genetic representations, this is straightforward. A distinction is first made between *cardinal* and *ordinal* genes. The alleles for ordinal genes are naturally ordered, as, for example, when the gene represents a continuous or contiguous variable, and in this case it makes sense to define a variable distance between alleles (normally the Euclidean distance). With cardinal genes there is no particular relationship between the various alleles for a given gene, so the discrete metric is used, making the distance between any pair of distinct alleles one. The distance between solutions is then computed simply by summing the distances between the gene values at each locus. For cardinal genes, this measure reduces to the familiar Hamming distance.

Allelic representations are a little more complex to handle. It is implicit in the definition of genes that every chromosome has the same number of alleles, because every gene has precisely one well-defined value for each solution (and there is no other source of alleles). In the case of allelic representations this need not be the case. For example, if the search space consists of sets of variable size, it would sometimes seem appropriate to take the (variable number of) elements of the set as alleles. A satisfactory approach for allelic representations is to define an overlap or similarity measure in the first instance by counting the number of alleles that are or are not common to the two solutions in question. A distance measure, D, between two allelic chromosomes can then be introduced, equal to the half the number of alleles present in only one of the two chromosomes:

$$D(X,Y) \triangleq (|X \setminus Y| + |Y \setminus X|)/2, \tag{5.15}$$

where $\setminus$ denotes set subtraction. Note that this is consistent with the straightforward definition of $D(x,y)$ for genetic representations.

Definition (Minimal mutations). A genetic or allelic chromosome y will be said to be a *minimal mutation* of x if and only if there is no other chromosome in $\mathcal{C}$ closer than y to x with respect to D. Thus the set $M_D(x)$ of minimal mutations of x is given by

$$M_D(x) = \{y \in \mathcal{C} \mid \forall z \in \mathcal{C} \setminus \{x\} : \; D(x,y) \leq D(x,z)\}. \tag{5.16}$$

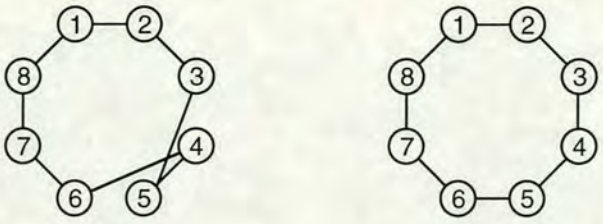


Figure 5.3: In the undirected edge representation, the left-hand tour is { 1–2, 2–3, 3–5, 4–5, 4–6, 6–7, 7–8, 1–8 } and the right-hand tour is represented by { 1–2, 2–3, 3–4, 4–5, 5–6, 6–7, 7–8, 1–8 }. In this representation, these two tours are minimal mutations of each other, because they differ by exactly two edges and no pair of distinct tours, in this representation, differ by fewer than two edges.

■

Notice that there is no suggestion that the minimal mutations should have distance 1 from the solution to be mutated; indeed, while this will clearly be the case for orthogonal genetic representations, it is the case for none of the four representations for the TSP considered in chapter 7.

Note also that given chromosomes x, y and z, it is not necessarily the case that $D(x,y) < D(x,z)$ implies that y can be generated from x in fewer minimal mutations than can z.

Informally then, minimal mutations are defined to be those moves which change the fewest possible number of alleles in a solution (in non-orthogonal representations it may be necessary to change more than one allele at a time to maintain legality). For example, in the undirected edge representation for the TSP (see chapter 7), any tour that can be constructed from another by reversing some section of it is one of its minimal mutations, because reversing a section involves breaking only two edges and there is no pair of tours that differ by only one edge (figure 5.3).

5.3.1 Binomial Minimal Mutation (BMM)

Binomial minimal mutation, BMM, a generalisation of standard point-wise mutation, has been proposed in Radcliffe & Surry (1994b). As defined there, BMM performs a binomially-distributed number of minimal mutations (parameterised by the genome length and a gene-wise mutation probability), and does not forbid mutations which 'undo' previous ones. A slightly modified definition is presented here based on the "effective chromosome length": loosely, the number of minimal mutations required to randomise the chromosome.

BMM can be defined for any representation—genetic or allelic—provided that the following conditions are met:

1. The distance $D(x,y)$ between two chromosomes x and y can be suitably defined. Typically $D(x,y)$ is taken to be half the number of alleles that are present in only

one of the two parents. (In the case of fixed-length chromosomes this is equal to the number of genes minus the number of shared alleles.)

2. It is computationally feasible to identify minimal mutations based on $D(x, y)$ as described above. Although not absolutely required, BMM is formulated primarily for the case in which the distance between a chromosome and each of its minimal mutations is the same for all chromosomes, and every chromosome has the same number of minimal mutations.

3. Any (legal) point in the representation space $\mathcal{C}$ can be reached by a finite sequence of minimal mutations from any other point (a sequence of minimal mutations is ergodic; Radcliffe, 1991a).

This last requirement can be tested by considering the probability distribution generated by applying a sequence of minimal mutations to arbitrary starting chromosomes

$$\mu(y) \triangleq \frac{1}{\|\mathcal{C}\|} \sum_{x \in \mathcal{C}} \lim_{k \to \infty} P\{\hat{\mathcal{M}}^k(x) = y\} \tag{5.17}$$

where $\hat{\mathcal{M}} : \mathcal{C} \longrightarrow \mathcal{C}$ is a stochastic operator that returns a randomly (uniformly) chosen member of $M_D(X)$. If ergodicity is satisfied, each term in the sum should be equal (independent of x), and if minimal mutations are unbiased we would expect $\mu \sim U(\mathcal{C})$.

By requiring ergodicity, repeated application of minimal mutations to a fixed starting chromosome provides a formal mechanism for generating random chromosomes. This leads to the notion of an *effective chromosome length*, ℓ_n, chosen so that a sequence of $O(\ell_n)$ random minimal mutations randomises the original chromosome (the arbitrary constant factor in ℓ_n will later be folded into the mutation probability, so will henceforth be ignored). Specifically, if n is the size of the problem instance, then we seek the minimal ℓ_n for which there is a constant $\varepsilon > 0$ such that

$$\forall n \to \infty \, \forall x, y \in \mathcal{C}_n \; : \; P\{\hat{\mathcal{M}}^{\ell_n}(x) = y\} > \varepsilon / \|\mathcal{C}\| \tag{5.18}$$

can be satisfied.

For orthogonal representations, the effective and actual chromosome lengths are typically equal[2], i.e. $\ell_n = n$, but when the representation is highly non-orthogonal (so that the likelihood of "undoing" previous minimal mutations is high), we may have $\ell_n > O(n)$. For example, with the Dedekind representation introduced in chapter 8, we find that $\ell_n = n^2$.

After introducing an additional parameter p_M, that specifies the probability of performing each possible minimal mutation, we are now in a position to define BMM.

[2]For example, consider the standard binary representation, where the minimal mutation is a bit flip. For a chromosome of length n, the expected value of an arbitrary bit after ℓ_n randomly located bit-flips can be shown to limit to $(1 \pm (1 - 2/n)^{\ell_n})/2$ (with the sign depending on the bit's initial value). It is then straightforward to deduce that the critical value for randomising the chromosome is $\ell_n = O(n)$.

First, a number k of mutations to perform is selected from the binomial distribution $B(\ell_n, p_M)$. This choice ensures that in the case of orthogonal representations BMM's behaviour mimics that of conventional gene-wise mutation. A sequence of k chromosomes is generated, each of which is a minimal mutation of the previous, the first in the sequence being the chromosome to be mutated and the last being the resultant chromosome. Thus BMM is simply the kth iterate of $\hat{\mathcal{M}}$:

$$\mathrm{BMM}(X, p_M) \triangleq \hat{\mathcal{M}}^k(X), \tag{5.19}$$

where

$$k \sim B(n, p_M). \tag{5.20}$$

Note that this operator does not exclude the possibility that subsequent mutations reverse earlier ones, but in practice the likelihood of this is low for small values of p_M. This implies that in the limit of orthogonal genes, conventional gene-wise mutation is essentially recovered. Although the operator could be redefined to ensure that no mutation in the sequence is accepted if it generates a chromosome already visited in the sequence of minimal mutations, this has been avoided as it is relatively troublesome to implement in general. For small values of p_M, and for representations in which the number of minimal mutations for each chromosome is large, the check makes little practical difference, and in situations where this is not the case, the effective length ℓ_n compensates for exactly this tendency by requiring a longer sequence of minimal mutations.

5.3.2 Representation-independent Hill-climbing and Memetic Algorithms

Given a move operator, which in the present case will be taken to be the minimal mutation operator, it becomes a simple matter to define a family of hill-climbing operators. A hill-climber can be obtained by repeatedly trying moves generated by the move operator, in some sequence, and accepting all those that improve upon the current solution. This process continues until none of the moves that the operator can generate improves the solution. Numerous strategies are possible for generating the moves, but experience suggests that the more random the order, the better will be the expected performance. The method used here generates moves in a pre-determined order for any particular hill-climbing, but this order changes each time the hill-climber is invoked. Davis (1991a) has argued that it is usually desirable to accept neutral moves (ones that have no effect on fitness), but this is not central to the current discussion.

The application of a local optimiser such as the hill-climber described always succeeds in generating a local optimum (with respect to minimal mutations). Following Radcliffe & Surry (1994c), which also contains further details of the hill-climbing techniques used, chapter 9 defines a *memetic algorithm* as a genetic algorithm in which

local optimisation is applied to all solutions before evaluation. This can be thought of as a genetic algorithm applied in the subspace of local optima, with local optimisation acting as a repair mechanism for children lying outside this subspace (i.e. not being locally optimal). Particularly when the evaluation function under consideration is decomposable—in the TSP, for instance, the length of a child tour similar to its parent can be computed more easily given the length of the parent tour—hill-climbing is relatively cheap, so memetic algorithms might be expected to perform relatively better than genetic algorithms for this problem.

Chapter 6

Constructing Domain-specific Evolutionary Algorithms

What, unless biological science is a mass of errors, is the cause of human intelligence and vigour? Hardship and freedom: conditions under which the active, strong, and subtle survive and the weaker go to the wall; conditions that put a premium upon the loyal alliance of capable men, upon self-restraint, patience, and decision.

(H.G. Wells)

The claimed motivation for separating problem representation from algorithm was that in addition to making explicit our hypotheses about problem structure, it enabled a much more mechanistic approach to new problem domains. In this chapter, we discuss this issue in more detail. In section 6.1 we discuss how formal algorithms (defined independently of any problem domain) can be instantiated with formal representations for any problem domain of interest. We stress that this instantiation process is one of mathematical derivation rather than direct computational implementation, so that the search strategy which is eventually implemented has simplified, problem-dependent form.

We go on, in section 6.2 to discuss the implications of this approach to performance prediction. We explore ways in which the performance of a problem-specific search strategy might be predicted given the mathematical definition of the formal algorithm and operators, along with suitable measurements of the representation. This leads to the ideas of forma variance, which are shown in later chapters to give good qualitative prediction of algorithmic performance.

Finally, in section 6.3 we summarise the advantages of overall approach presented in the last three chapters, and summarise potentially fruitful avenues for future exploration.

6.1 Formal Algorithms + Formal Representations = Search Strategies

In order to construct a practical search strategy for a given problem domain, we simply combine a formal algorithm with an appropriate representation of that problem domain. There is no need to construct new move operators, as we simply instantiate those defined in the formal algorithm of choice (obviously, in some cases, highly problem-specific operators may be advantageous). Since *exactly* the same formal algorithm (for example, that shown in chapter 5) can be instantiated for two different representations (of either the same or different problem domains), one can make much more definite statements about the quality of the algorithm itself (as it is defined independently of any problem). It is also possible to fix the representation and vary the algorithm, allowing more meaningful comparisons between algorithms.

For several of the representations shown in table 4.1 (chapter 4), the generalised operators defined in chapter 5 reduce to traditional forms. For example, for any orthogonal representation, R^3, RTR and RAR all reduce to uniform crossover (as defined by Syswerda, 1989), GNX reduces to N-pt crossover, and BMM becomes simple gene-wise point mutation. For the Dedekind real representation, introduced in chapter 8, we show that R^3, RTR and RAR reduce to blend crossover with parameter $\alpha = 0$, as defined by Eshelman & Schaffer (1992) and widely used in evolution strategies (Bäck *et al.*, 1991); and that BMM is equivalent to gaussian creep mutation (Surry & Radcliffe, 1996c). As shown in chapter 7, if we consider the undirected edge representation for the travelling sales-rep problem, RAR becomes a variant of edge recombination (Whitley *et al.*, 1989) and R^3 reduces to a "weaker" variant of the same operator. BMM here involves a binomially distributed number of sub-tour inversions, whereas for the city-position representation, BMM reduces to a binomially distributed number of city exchanges (Radcliffe & Surry, 1994b).

Such reductions imply that formal algorithms defined using these operators reduce to commonly used search strategies in the relevant problem domains. To illustrate this, the algorithm shown boxed on page 62 is instantiated in the box below for both the travelling-sales rep problem using the undirected-edge representation, and for a real-parameter function optimisation problem using the Dedekind representation. This results on the one hand on a strategy based on edge recombination and sub-tour inversions, and on the other in one based on blend-crossover and gaussian creep-mutation. Both of these strategies have been widely used in their respective domains, but it was not clear before now that they could be viewed as exactly the same formal algorithm.

Search strategy as algorithm plus representation		
Problem domain:	**TSP**	**Real-parameter opt.**
Representation:	**Undirected-edges**	**Dedekind**
Choose initial population:	of random tours	of random vectors
Evaluate each solution:	by measuring tour length	using provided $f(\boldsymbol{x})$
Select two parents:	using (deterministic) binary-tournament selection	
Recombine parents using:	variant of edge-recomb.	BLX-0 (see figure 8.2)
Mutate the child with :	binomial number of sub-tour inversions	gaussian creep-mutation for each parameter
Evaluate, replace worst:	if the child does not exist in the population	
Repeat:	until termination criterion	

Note that both the representation and algorithm are mathematical constructions and need not be directly related to the actual way in which the data structures and computer code for the resulting search strategy is implemented on a digital computer[1]. Thus, rather than simply plugging together different bits of computer code, we plug together different bits of mathematics from which we can formally *derive* an actual implementation in a well-specified way. For example, the (formal) Dedekind representation for real numbers has (in the limit) an infinite number of genes, yet it is a simple matter to mathematically derive forms of the various operators suitable for (finite!) implementation. The distinction between mathematical derivation and computational implementation is graphically illustrated in section 6.2.2.

6.2 Algorithmic Performance prediction

As discussed in section 3.4, there is considerable freedom in choosing exactly how to measure the performance of evolutionary algorithms. For example, it would be possible to choose to consider rate of convergence, time to solution, or robustness of results, off-line or "best seen" performance, number of evaluations or wall-clock time in almost any combination. Moreover, robustness is an issue not only with respect to differently seeded stochastic runs, but also as a function of different problem instances, perhaps of quite different complexities, and it may be that different relative performances will be achieved by different representations as the parameters of the algorithm, the operators used and the particular problem instance are varied.

Despite this panoply of choices, it seems reasonable to expect that for at least some classes of problem there will be a broad congruence of results over many of these different

[1]Thus the title of this section has been inspired by but differentiated carefully from the prior works by Wirth (1976) and Michalewicz (1992).

measures. For other problem classes it might be possible to rank the performance of different representations with respect to a particular performance measure and some chosen set of parameters and operators.

The principal aim of this section is to identify and measure properties of representations that are well-correlated with their performance in evolutionary search. If this could be achieved it would both allow some level of performance prediction and increase understanding of the search techniques themselves.

We begin by suggesting an analogy between evolutionary optimisation and machine learning algorithms, in section 6.2.1. This enables us to view the performance of an evolutionary algorithm in terms of its success in forming correct (though implicit) hypotheses about the structure of the problem domain. The hypotheses themselves are couched in terms of the formae generated by the chosen representation. This is emphasised by showing that an unbiased representation leads directly to an unbiased (random) search algorithm in section 6.2.2, and leads to the discussion in section 6.2.3 of the utility of forma variance as a prospective performance indicator. The process is illustrated analytically for a trivial subset problem in section 6.2.4, prior to its utilisation in later chapters for more practical problem domains.

6.2.1 A Machine-learning Perspective

We gain an interesting perspective on the problem of relating representation to algorithmic performance from the machine learning literature (see, for example Mitchell, 1997). In this field, a learning algorithm explicitly selects between competing hypotheses based on training data in order to generalise to unseen data. In the simplest case, the (arbitrary) concept to be learned is simply a boolean function defined over all possible test-cases, and every hypothesis considered by the learning algorithm is just a subset of the the possible test-cases for which the concept is supposed to be true. It is easy to argue that in order for a learning algorithm to generalise effectively, it must restrict itself to consideration of a limited number of hypotheses from which to identify the target concept. That is, an *inductive bias* must be imposed externally. (Intuitively, if the set of all possible hypotheses—the power set of the set of potential input data—were considered, then many hypotheses would be consistent with any non-exhaustive sample of observed data, leading directly to results which parallel the "No Free Lunch" theorems of chapter 3, e.g., Mitchell, 1980.)

To establish the parallel with evolutionary (or other optimisation) algorithms, we simply view such an algorithm as using the function values for points previously visited in the search space to infer the likely location of high-quality points not yet visited. Every time the algorithm selects a new point to sample, it bases its choice on its current (implicit) hypotheses about the relationship of solution structure to function value.

Having carefully constructed formal algorithms that manipulate solutions based solely on the forma membership properties implied by the chosen representation, we argue that any such hypothesis must be expressible in terms of formae. (In fact, we explore the consequences of using an unbiased representation in section 6.2.2 to illustrate this point.)

This sheds new light on the principle of minimal alphabets, which states that

> *[t]he user should select the smallest alphabet that permits a natural expression of the problem.*
>
> (Goldberg, 1989c)

This principle is motivated by controversial considerations of implicit parallelism (Holland, 1975) which are sometimes argued to suggest inherent superiority for lower cardinality (e.g. binary) representations. Although this principle is not universally accepted as helpful in the construction of representations (e.g. Antonisse, 1989; Davis, 1991b; Radcliffe, 1992b), it still appears to generate substantial confusion, typically because the requirement for "natural expression" is too often ignored.

In the language of machine learning, it is clear that by increasing the generality of hypotheses considered by the algorithm, we are more likely to cover arbitrary target concepts. However, it becomes correspondingly more difficult to select correctly between the competing hypotheses. This appears to be the point missed by the principle of minimal alphabets—it is definitely not in general the case that "greater information becomes available using smaller alphabets" as Goldberg (1990) states. Although the representation becomes more expressive (in the sense of being able to represent a larger number of subsets of $\mathcal{S}$—though even this is disputable; Antonisse, 1989), this comes *at the expense* of making the actual structure present in the problem more difficult to exploit (precisely because there are more subsets to choose between). After all, in the absence of definitive knowledge about problem structure, the additional real information obtained from sampling a previously unvisited point is clearly unaffected by how that point is represented. In Goldberg's own words, using a binary alphabet means that "many hypotheses can be formulated regarding the association between substring values and high fitnesses", but what he fails to mention is the difficulty in correctly determining which of these hypotheses are actually correct. Rather than a minimal alphabet what is wanted is an *expressive alphabet* suitable to the problem domain at hand.

6.2.2 A Representation for Unbiased Search

We have claimed that by carefully choosing a characterisation of the search domain (which captures features of solutions relevant to performance), we are able to generate representations and subsequently problem-specific search strategies whose bias is

directly related to our stated beliefs about problem structure. In order to emphasise this point, we will consider a representation generated by a manifestly unbiased characterisation of a problem domain, and see that it results in an unbiased search algorithm.

Using forma analysis, we represent arbitrary elements of the search space by identifying to which equivalence classes they belong. These equivalence classes are simply subsets of $\mathcal{S}$ which are believed to group together solutions with related performance. It is perfectly feasible, mathematically, to identify every possible subset of $\mathcal{S}$ as an equivalence class. This reflects an indifferent prior in the Bayesian sense; that any subset is as likely as any other to contain solutions of related performance. To do this, we characterise the problem domain using equivalence relations of the form

$$\psi_i(a,b) = \begin{cases} 1, & a,b \in \mathcal{S}_i \text{ or } a,b \notin \mathcal{S}_i \\ 0, & \text{otherwise.} \end{cases} \tag{6.1}$$

where $\mathcal{S}_i$ is the ith element of $\mathbb{P}(\mathcal{S})$, the power set of $\mathcal{S}$, under an arbitrary enumeration. This in turn generates $2 \cdot 2^{|\mathcal{S}|}$ equivalence classes, of the form

$$\xi_i^0 = \mathcal{S}_i, \xi_i^1 = \mathcal{S} \setminus \mathcal{S}_i \tag{6.2}$$

leading to a representation with $2^{|\mathcal{S}|}$ binary genes, each indicating whether the solution does or doesn't belong to the ith element of $\mathbb{P}(\mathcal{S})$.

To make things concrete, consider an example with $\mathcal{S} = \{1,2,3\}$ and the two solutions 1 and 2. We then have:

Gene:	$\varnothing$	$\{1\}$	$\{2\}$	$\{3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2,3\}$
$\rho(1)$	0	1	0	0	1	1	0	1
$\rho(2)$	0	0	1	0	1	0	1	1

Although this is clearly a representation for which it is infeasible to physically store or manipulate chromosomes (see section 4.1.1, page 47), we find that upon instantiating our representation-independent operators we are left with simple, computable operations. We examine both mutation and recombination operators in this context.

Consider BMM: we must first determine the minimal mutants of a solution a represented by the chromosome x. We argue that every solution $b \in \mathcal{S} \setminus \{a\}$ is a minimal mutant of a. To see this, note that a and b will have identical values for each gene i such that $a,b \in \mathcal{S}_i$ or $a,b \notin \mathcal{S}_i$; and opposite values for each gene j such that $a \in \mathcal{S}_j, b \notin \mathcal{S}_j$ or vice versa. Because we are considering *every* subset of $\mathcal{S}$ when classifying these $\mathcal{S}_i$ and $\mathcal{S}_j$, it is clear that the number of sets $\mathcal{S}_j$, and hence the number of genes at which the two chromosomes differ, must be constant, independent of the choice of a or b. Therefore, every $b \neq a$ is a minimal mutant of a.

A sequence of minimal mutations then amounts to a random walk on $\mathcal{S}$, with each step moving with equal probability to any other point in $\mathcal{S}$. This effectively results in BMM reducing to random sampling.

Moving on to recombination operators, consider first R^3 and RTR. Given two parents, a and b, a child generated by R^3 must contain all alleles present in both parents, and one generated by RTR must contain only alleles present in either parent. By observing that $\rho(a)$ and $\rho(b)$ both contain the allele meaning "in $\{a, b\}$", it is clear that both $\mathrm{RTR}(a, b)$ and $\mathrm{R}^3(a, b)$ simply generate a child equal to one of the two parents.

The analysis of RAR_w is slightly more complicated. Recalling the definition in section 5.2.3 (page 66), we construct a proto-child by repeatedly adding alleles (if compatible) from those contained in the parents, with shared alleles chosen with weight w and unique alleles chosen with weight 1.

Now, each element of $\mathbb{P}(\mathcal{S})$ can be represented uniquely as one of X, $X \cup \{a\}$, $X \cup \{b\}$ or $X \cup \{a, b\}$ for some $X \in \mathcal{S} \setminus \{a, b\}$. Note that $\rho(a)$ and $\rho(b)$ have identical alleles in the first and last case (0 and 1 respectively), and opposite alleles otherwise. If an allele of the first or last form (shared by both parents) is added to the proto-child, a randomly selected subset of $\mathcal{S} \setminus \{a, b\}$ will be excluded from the set of admissible children. There are two such alleles for each set X (a zero indicating neither parent is in X and a one indicating both parents are in $X \cup \{a, b\}$). If an allele where the parents disagree is chosen, a randomly selected subset of $\mathcal{S} \setminus \{a, b\}$ will be excluded from the set of admissible children *along with* one of a or b. There are four such alleles for each set X, two excluding a and two excluding b.

We see that the likelihood of excluding a solution $c \notin \{a, b\}$ from the set of admissible solutions at each step is $1/2$ but the likelihood of excluding a (or equivalently b) is only $2/(4 + 2w)$. For any $w > 0$ then, a (and b) is more likely to remain within the set of admissible solutions than any other specific solution. Thus, although $\mathrm{RAR}_w(a, b)$ can generate any child in $\mathcal{S} \setminus \{a, b\}$ with equal likelihood (for example, by selecting the allele "in $\{a, c\}$" from a and "in $\{b, c\}$" from b), it is more likely to generate one of the parent solutions (with certainty as $w \to \infty$).

However, it remains the case that any formal algorithm incorporating BMM and one of the recombination operators will, when instantiated with this representation, simply result in an (unbiased) random search. This is to be expected given our initially unbiased characterisation of the search domain—no *formal* operator can distinguish members of $\mathcal{S}$ not equal to a or b using this representation—but nevertheless provides a compelling example of how the formalism we advocate transforms our explicitly stated hypotheses about solution structure directly into an algorithm incorporating that same bias.

6.2.3 Forma Variance as a Performance Indicator

A primary aim of this work is to find measurable properties of representations that are correlated with the performance of an evolutionary algorithm. This is important in order to facilitate choosing between alternative representations for a given problem domain. The development of forma analysis, which we use to generate representations, was based a set of observations about the schema theorem (Holland, 1975) in its generalised form (Radcliffe, 1991a; Vose & Liepins, 1991b). These suggest that representations based on formae (generalised schemata) with lower fitness variance would be expected—other things being equal—to allow more effective search than those based on formae with higher fitness variance. Fitness variance of formae is thus a natural candidate for a measure to act as a predictor of representation performance.

Consider first those formae whose members in the current population are fitter than average. Along with the chromosomes that instantiate them, these formae are selected for reproduction more often, or selected for deletion less often, than formae with lower observed performance. Consider now the effects of applying genetic operators. If the recombination operator used respects the representation, this will ensure that whenever two parents are recombined their child will instantiate all formae of which they share membership. If the recombination operator is not very disruptive, even when only a single parent is a member of such formae the probability of generating new instances of them is relatively high. If the recombination operator is non-respectful, the extent to which these arguments apply depends on the degree to which respect is violated. Similarly, for (typical) low mutation rates, a child produced by mutation will share membership of most formae of low to medium order with its parent. Finally, although hill-climbing from a random starting point will usually produce a chromosome very different from its parent, when the parent is in the vicinity of a local optimum (with respect to that hill-climbing operator) as will tend to be the case after recombination in an effective memetic algorithm, it is likely that hill-climbing will not change a very large number of alleles, so forma membership is still to some extent preserved.

Turning now to forma construction—i.e. the sampling of chromosomes in formae of which the parents are not instances—observe that such newly sampled formae are likely to overlap in their allele composition with formae currently seen to be performing well, particularly to the extent that the recombination operator used is respectful. This is important because the way in which evolutionary (and particularly genetic) search is thought to proceed is by sampling smaller (higher order) formae that are intersections (compositions) of larger (lower order) formae of high relative fitness (the spirit at least of the "building block" hypothesis of Goldberg, 1989c). Thus recombination is thought of as gradually building up complex solutions by combining "fit" components. While challenging search problems will exhibit some possibly large degree of non-linearity,

so that combining components that are observed to be (relatively) fit will not always result in ever-fitter individuals, this approach retains some validity.

If the recombination operator used is transmitting, it can further be observed that alleles common to chromosomes and formae exhibiting above average performance at the current time step are also preferentially selected, both individually and collectively. The essence of the notion of assortment is that assorting recombination operators are capable of bringing together any compatible (non-competing) formae from the parents in a child. This seems to be essential given the model of genetic (and more widely evolutionary) search considered, the wide-spread use of non-assorting recombination operators for non-orthogonal representations notwithstanding. Needless to say, in the presence of selective pressure towards better solutions and formae, the new formae of which instances are created tend to be intersections of fitter, larger (lower order) formae.

To provide further motivation for these ideas, we borrow from the theory of global random search (Zhigljavsky, 1991). First note that all of the formal genetic operators defined in chapter 5 generate children according to a probability distribution over $\mathcal{S}$ concentrated "near" the parent solution(s), where near is defined by some measure of shared forma membership. (For example with a respectful recombination operator it is non-zero only over the *similarity set* of the parents; Radcliffe, 1994.) This (probabilistically) biases the child towards formae containing previously sampled points with high fitness. To make this an effective strategy, such children must also tend to have higher fitness than random solutions, which implies that we must be able to estimate the fitness distribution within the region of interest more tightly than that over the whole search space (at least in some cases).

In his work on global random search for real-parameter problems, Zhigljavsky introduces the *prospect* function, which is trivially extensible to arbitrary search domains. This is a sub-additive function defined on subsets of the search domain, $\lambda : \mathbb{P}(\mathcal{S}) \longrightarrow \mathbb{R}$, which indicates the likelihood of finding the optimum within the specified subset. The prospect function is then used to guide future sampling, for example choosing between two competing sets A and B by comparing $\lambda(A)/|A|$ and $\lambda(B)/|B|$.

With perfect knowledge, we could calculate the true prospect function, $\lambda(A) = 1$ if $a^\star \in A$, and 0 otherwise (taking the simple case of an unique global optimum $a \in \mathcal{S}$). However, in practice, we must use some proxy, based on sampled information about the subset A. Zhigljavsky suggests several alternatives, including estimating $\text{mean}_A f$, $\min_A f$ or $\max_A f$; establishing confidence intervals on $\max_A f$; or estimating the likelihood that $\max_A f > \max^\star_A f$ ($\max^\star$ being over already observed points in A). Clearly all of these estimates will be improved if the actual distribution of f over A is more compact (has lower variance).

Though somewhat imprecise, these arguments suggest that there are important senses in which evolutionary algorithms are directed by observed forma fitnesses, especially when recombination plays a major rôle, and when the recombination operators are (more strongly) respectful, transmitting and assorting. The schema theorem also points to the important rôle observed forma fitnesses play in guiding the search. Given this, the distribution of fitnesses within formae would seem to be central to how search proceeds. (This of course makes the assumption that the formae under consideration are relevant to the search process—although this may be clear for search strategies derived directly from formal representations and algorithms, it is less clear how relevant formae can be inferred given only an algorithm designed from scratch for a particular problem domain.) It is therefore worth considering the distributions that might be expected or desired.

First, it seems clear that for any challenging search problem large formae will have wide distributions of fitnesses, while if the search is to be guided by forma fitnesses it would be helpful if smaller formae had narrower distributions. Indeed, if formae were simply random collections of solutions from the search space it would be impossible to collect any useful directional information from collecting fitness samples from formae. Ultimately, of course, the interest is really in the fitness of the fittest member of each sampled forma, but unfortunately that is unavailable without exhaustively searching each forma. It is therefore necessary to restrict consideration to statistical measures available from sampling formae.

We might expect then that the variance and other moments of formae from any representation in which the genes carry useful fitness information would be a maximum for large formae and fall towards zero for very small formae. More importantly, however, it might be further expected that representations for which fitness variance tends to fall more quickly as a function of increasing forma size would give evolutionary search algorithms stronger and more exploitable information than those with broader fitness distributions.

Early work in this area was carried out by Hofmann (1993), who measured fitness variance of formae in the travelling sales-rep problem (although he did not employ a fully representation-independent algorithm). He compared instantiations of random assorting recombination (RAR; Radcliffe, 1994) using various representations, and the *strategic edge crossover,* SEX, developed by Moscato & Norman (1992) as an extension of edge recombination (Whitley *et al.*, 1989).

This is extended by the experiments reported in later chapter, which measure fitness variance and various other fitness moments for formae in different representations for the same problem domain. These are seen to provide good qualitative indications of the relative performance of a fixed formal algorithm instantiated with the alternative

representations.

6.2.4 Illustration: Trivial Subset Selection ("One Counting")

Consider the trivial subset selection problem in which we seek some optimal subset of a universal set U where the fitness of a candidate subset is simply its cardinality. That is, $f(a \subseteq U) = |a|$. If we characterise this problem using equivalence relations of the form

$$\psi_{ij}(a,b) = \begin{cases} 1, & \text{if } i,j \in a,b \\ & \text{or } i,j \notin a,b \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

(where i,j are distinct elements of U), the resulting representation has orthogonal binary chromosomes of length $|U|$ with the ith gene indicating whether on not the ith element of U is present in the candidate subset. The fitness of a chromosome x representing the solution a is then simply given by the number of 1's in its genome. Although not a practically interesting problem (indeed a simple hill-climber is a more efficient solution technique than any evolutionary algorithm), it provides a convenient example whose forma variance properties can be derived analytically.

Formae for this problem can be described by indicating which genes have fixed allele values, and which are arbitrary. For instance, with $n = 5$, the forma described by 1□01□ consists of all subsets that contain the first and fourth elements of the universal set, and don't contain the third element. The fitness distribution of this forma has a fixed contribution from the number of 1's in its description, plus a variable contribution depending on how the "don't care" symbols are replaced. Indeed, it is clear that for any forma ξ with order (number of defining positions) $o(\xi)$, the distribution of fitnesses can be written

$$f(\xi) \sim c_\xi + B(n - o(\xi), 0.5) \tag{6.4}$$

where $n = |U|$, $B(n,p)$ is the binomial distribution and c_ξ is the (constant) number of 1's in the description of ξ. It is straightforward to derive the mean and variance of fitnesses in ξ as:

$$\overline{f}_\xi = c_\xi + \frac{n - o(\xi)}{2}, \tag{6.5}$$

$$\mathrm{var}_\xi f = \frac{n - o(\xi)}{4}. \tag{6.6}$$

This is shown graphically in figure 6.1. Note that in this simple example the forma variance (or equivalently standard deviation) is homogeneous—it is dependent only on the size (order) of the forma, and not on, for example, its mean fitness (in contrast to the more difficult problems studied in chapters 7 and 8. This indicates that we can estimate the prospect function equally well throughout the search space (in fact, using an estimated mean fitness, we can form tight bounds on the fitness values within a forma ξ given that the distribution is known to be binomial).

6.3 Discussion

This chapter has presented the culmination to a more formal approach to evolutionary search, based on separating a search strategy into a representation and an algorithm. We have introduced a disciplined methodology for attacking new problem domains—instead of simply using evolutionary "ideas" to invent new operators, one need only provide a *characterisation* of the problem that explicitly captures beliefs about its structure, and then instantiate an existing algorithm with the derived representation. This applies equally to problems with non-orthogonal representations where traditional evolutionary algorithms are inapplicable. We have demonstrated, by way of example, that identical algorithms can be applied to both the TSP and real parameter optimisation, yielding familiar (but apparently quite different) concrete search strategies.

Because these formal algorithms are independent of any particular representation, it is possible to transfer them to arbitrary problem domains, and to make meaningful comparisons between them. By making the rôle of domain knowledge more explicit we are also directed to more reasoned investigation of what makes a good representation for a given problem.

In addition to choosing between representations, the framework leaves scope for improving (problem-independent) formal algorithms. Although we know from "no free lunch" considerations that all algorithms which do not revisit points in the search space are isomorphic in the absence of domain knowledge, practical algorithms are generally not non-revisiting. This implies that we might improve formal algorithms by changes which affect the probability with which they revisit points. For instance, a formal evolutionary algorithm which enforces uniqueness in its population might be better averaged over "all" problems than one which did not. Similarly, it might be possible to determine optimal problem-independent values for algorithm parameters, such as the μ/λ ratio in evolution strategies (Bäck *et al.*, 1991). We might also study how to improve a formal algorithm for representations which possess some particular properties.

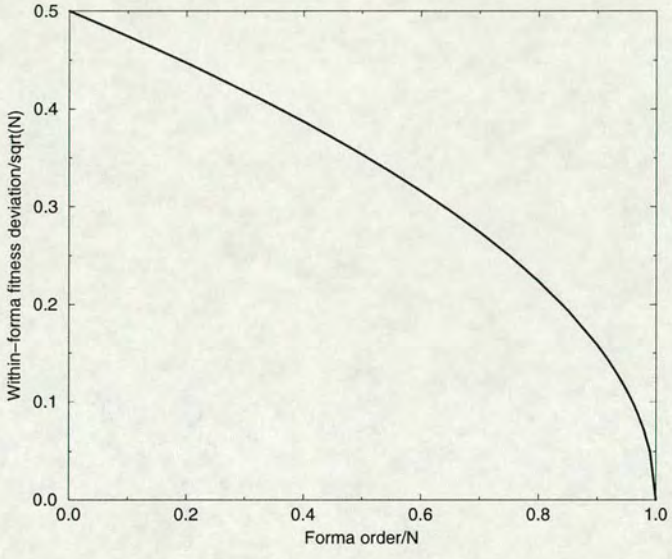


Figure 6.1: The graph shows the standard deviation of fitness as a function of forma size for the trivial subset-selection problem (familiar to many as "one-counting"). In this simple problem, forma variance is dependent only on the order of the forma (linearly; or logarithmically on its size) and not on any other structure (e.g. mean fitness). This indicates that we can estimate the *prospect function* equally well throughout the search space.

Chapter 7

Case Study: Travelling Sales-rep Problem

> *The Darwinian mechanism of vary-and-select, vary-and-select has one enormous difference from the process of design. It operates by hindsight rather than foresight. Evolution is always away from known problems rather than toward imagined goals. It doesn't seek to maximize theoretical fitness; it minimizes experienced unfitness. Hindsight is better than foresight.*
>
> (Stewart Brand)

The primary goal of this chapter is to make concrete our exploration of representation in the context of evolutionary search. The particular problem domain on which we will focus is the travelling sales-rep problem (TSP), and results will be presented for preliminary empirical studies of this problem. After describing the TSP in section 7.1, we proceed in section 7.2 to formalise four alternative representations which can be applied to this problem domain. The characteristics of these representations are studied and the instantiated for the genetic operators defined in chapter 5. Following the discussions of chapter 6, we study the utility of forma variance as a predictive tool for algorithmic performance in section 7.3, with experimental results presented in section 7.4. These results confirm a strong correlation between observed forma variance and algorithmic performance.

7.1 The Travelling-salesrep Problem

In the present chapter, we are interested in the problem domain, $\mathcal{D}$, comprising all (symmetric) travelling sales-rep problems. More specifically, $\mathcal{D}$ is the set of all problem instances in the class, where each problem instance takes the form of the search space, $\mathcal{S}$, to which it gives rise, along with the related cost function f. A problem instance consists of a particular collection of cities, so that the search space $\mathcal{S}$ is the set of all possible paths that visit every city exactly once—the set of all possible tours—and the aim is to find the shortest with respect to some metric. More precisely still, since there is no interest in the $2n$ equivalent paths through n cities that arise from the freedom

to choose the starting city and the direction of travel, $\mathcal{S}$ is the set of all *non-equivalent* tours. For example, given a set of four particular cities labelled 1 to 4, chosen for illustration to sit at the corners of a square,

$$\mathcal{S} = \left\{ \begin{array}{ccc} 4 & 3 & \\ 1 & 2 & \end{array}, \begin{array}{ccc} 4 & 3 & \\ 1 & 2 & \end{array}, \begin{array}{ccc} 4 & 3 & \\ 1 & 2 & \end{array} \right\}. \tag{7.1}$$

One convenient way to represent a tour is by listing the sequence of city labels in the order in which they are visited, so that the first tour shown in the set might be represented by 1234. This is called the *permutation representation,* denoted π.

It is important to be able to distinguish between the representative of a solution under some representation (the formal chromosome, or genotype) and the solution itself (the phenotype). For a particular representation ρ, the set of all chromosomes in this representation will be denoted $\mathcal{C}^\rho$. Given a chromosome $x \in \mathcal{C}^\rho$, and the solution $a \in \mathcal{S}$ to which it corresponds, the notation x^ρ will be used to mean "the solution represented by x in representation ρ". Thus x^ρ is a member of $\mathcal{S}$, and in the current example $x^\rho = a$. It is thus accurate to write

$$\mathcal{S} = \{1234^\pi, 1243^\pi, 1324^\pi\}. \tag{7.2}$$

In general, a TSP over n cities has an associated search space of size $n!/2n = (n-1)!/2$, arising from the $n!$ different permutations of the city labels and the $2n$ equivalent forms for any tour. In this chapter, tours shown in the permutation representation will be shown starting with city 1, followed by the lower-numbered of city 1's two neighbours.

7.2 Representations for the TSP

In the empirical tests that follow, four representations of the TSP will be used. The permutation representation π has been introduced above and is a (formal) genetic representation. Additionally, we will introduce an (allelic) undirected edge representation u (section 7.2.2), a (genetic) directed edge representation d (section 7.2.3), as well as a genetic variation c of the undirected edge representation (section 7.2.4), that uses compound genes and is equivalent to the corner representation suggested by Hofmann (1993). These four representations are then characterised in sections 7.2.5–7.2.7.

7.2.1 The Permutation Representation

The permutation representation is a formal genetic representation, and has already been introduced in section 7.1. Here the ith gene identifies the ith city in the tour. Thus

$$1324^\pi = \begin{array}{ccc} 4 & 3 & \\ 1 & 2 & \end{array}. \tag{7.3}$$

In the terms of chapter 4, we characterise the problem domain using basic equivalence relations that group together tours in which a specified city is visited at a prescribed position in the tour. As noted above, this characterisation arises more because this is perhaps the most natural way in which tours are written down rather than from any underlying belief about the structure of the cost function. Composition of these basic equivalence relations lead to formae that contain tours in which particular cities are visited at specified positions in the tour (possibly familiar to most readers as the o-schema of Goldberg & Lingle Jr, 1985). For example, in our four city example, the forma containing all tours that have city 2 in their second position would be given by

$$\xi' = \{1234^{\pi}, 1243^{\pi}\} \tag{7.4}$$

(corresponding to the o-schema 12$\square\square$), with the description set

$$\langle \xi' \rangle = \{(1,1),(2,2)\}. \tag{7.5}$$

Here the first gene has been fixed to have allele 1 in order to reduce degeneracy, so is formally redundant but this will not prove problematical. A complication does arise, however, from the freedom to choose the direction in which a tour is traversed. While this degeneracy can be formally removed with the convention introduced previously (always choosing the second city to have a lower numbered label than the nth), this is not wholly satisfactory. This problem is discussed in section 7.2.6.

We can calculate the size of a forma, $|\xi|$ directly from its order as

$$|\xi| = (n - o(\xi))!, \tag{7.6}$$

confirming that $|\xi'| = (4-2)! = 2$. (For formae of order 1 and specialised formae with order two, we must introduce a divisor of two in the above from symmetry considerations.)

The RAR^{π} and GNX^{π} operators discussed in chapter 5 are easy to implement for this representation, while the definition of BMM^{π} depends on the observation that the minimal mutation of a permutation is generated by exchanging the positions of two cities. An example of such an exchange would be

$$143526^{\pi} \overset{\mathrm{BMM}^{\pi}}{\longmapsto} 153426^{\pi}, \tag{7.7}$$

by exchange of genes 2 and 4. Minimal mutations are at distance 2 from their parents in this representation. (In terms of the number of edges broken, this corresponds to a "four change".) It is easy to see that all permutations can be generated by a sequence of such city exchanges, so the conditions for BMM are satisfied.

Gene transmission is incompatible with assortment in this representation, as the reader will be able to verify by trying to construct a member of the forma described by $\{(3,3),(5,2)\}$ from parents 123456^{π} and 134526^{π} without violating transmission.

7.2.2 The Undirected Edge Representation

When we consider the structure of the cost function for the TSP, it is clear that it is the connections between cities—the edges contained in the tour—which determine its cost. This suggests that a characterisation of the problem domain based on edges might be more effective than the permutation representation. In the *undirected edge representation*, denoted u, we choose the characterisation χ_u that identifies subsets of tours having a link (edge) between any two specified cities. For example, the tour 1324^π contains the (undirected) edges 13, 23, 24 and 14, so that

$$1324^\pi = \{13, 23, 24, 14\}^u. \tag{7.8}$$

Any solution can then be represented by the set of undirected edges that it contains (as above), as χ_u generates formae comprising all tours containing a specified set of edges. (Note that in this case χ_u does not generate equivalence relations.) For example, the forma ξ' consisting of those tours that contain the 12 edge is

$$\xi' = \{1234^\pi, 1243^\pi\} = \{\{12, 23, 34, 14\}^u, \{12, 24, 34, 13\}^u\}, \tag{7.9}$$

which is conveniently described by

$$\langle \xi' \rangle = \{12\}. \tag{7.10}$$

Note that the representation is allelic because genes are not directly defined—a solution is simply a collection of alleles. (As noted in chapter 4 however, we could recover a genetic representation by defining a basic equivalence relation for each of the $n(n-1)/2$ edges a tour could contain—generating a binary chromosome with $n(n-1)/2$ genes—but the actions of the operators would then tend to be dominated by the many edges *not* contained by the tour. See also Fox & McMahon, 1991 where this representation is employed.)

In order to calculate the size of general formae, $|\xi|$, we introduce the notion of *threads*. A thread is a group of two or more cities which are connected by edges contained in $\langle \xi \rangle$. If we denote the number of threads contained in ξ as $t(\xi)$, then it is clear that $t(\xi) \leq \lfloor n/2 \rfloor$, and that $t(\xi) \leq o(\xi)$. For instance, the forma ξ' defined above has $t(\xi') = 1$, and $o(\xi') = 1$. If we further define $t_0(\xi)$ to be the number of cities not present in ξ (so that $t_0(\xi') = 2$), we can see that

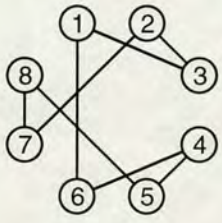
$$|\xi| = (t(\xi) + t_0(\xi) - 1)!\, 2^{t(\xi)-1} \tag{7.11}$$

as any tour contained in ξ can be constructed by stitching together all of the threads present in ξ along with all the cities missing from it in an arbitrary order, with each thread allowed arbitrary direction. Thus we confirm that $|\xi'| = 2!/2 \cdot 2^1 = 2$.

We now consider instantiating the representation-independent genetic operators RAR, GNX and BMM for the undirected edge representation.

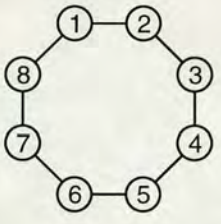
RAR^u, while somewhat harder to implement than for the permutation representation, follows straightforwardly from its definition, and the factor of two that is problematical for the permutation representation does not arise since the representation makes no reference to the direction in which a tour is traversed.

RTR^u reduces to a different operator because transmission is incompatible with assortment in this representation—to see this, observe that generating an instance of the forma described by $\{24, 23\}$ from parents 123456^π and 124356^π is impossible without violating transmission. (In general, solutions are written in the permutation representation in this chapter, even when they are being manipulated with respect to other representations.)



Parent 1

Unordered:	{ 7–8,	2–7,	1–6,	8–5,	4–5,	1–3,	6–4,	2–3 }
Ordered around tour:	(1–3,	3–2,	2–7,	7–8,	8–5,	5–4,	4–6,	6–1)
Aligned "Chromosome":	(3,	2,	7,	8,	5,	4,	6,	1)



Parent 2

Unordered:	{ 1–2,	6–7,	7–8,	1–8,	6–5,	3–2,	4–3,	4–5 }
Ordered around tour:	(1–2,	2–3,	3–4,	4–5,	5–6,	6–7,	7–8,	8–1)
Aligned with parent 1:	(1–2,	3–4,	2–3,	7–8,	8–1,	5–6,	4–5,	6–7)
Aligned "Chromosome":	(2,	4,	3,	8,	1,	6,	5,	7)

Figure 7.1: In order to use GNX, which requires a *genetic* representation, with the undirected edge representation, which is *allelic,* a special form of alignment must be performed, creating a "pseudo-genetic representation". Each parent is re-ordered so the the order and sense of the edges are taken by following the tour around. The second parent is then further re-ordered to align it with the first. "Pseudo-genomes" that GNX can then manipulate are then created. Notice, however, that the sense of an edge may be reversed by GNX when it inserts it in the child tour.

Because χ_u does not generate genes, it is not immediately clear that GNX is applicable for the undirected edge representation. However, it is possible to derive a problem-specific form of the operator by constructing an associated *pseudo-genetic representation*. This is achieved by arranging the edges in the order and sense in which they are encountered following the tour they describe in an arbitrarily chosen direction. The resulting list of edges will have every city exactly once as the "first" end of an edge, and can thus be used for alignment, allowing application of GNX^u as shown in figure 7.1. This borrows from the original view provided in Holland (1975) of crossover

as a locus-independent operator in the context of re-linking operators such as inversion.

Although it may seem as if this procedure simply manipulates the edges as directed, it is important to realise that this is not the case. For while the chromosome resulting from this process is the same as that used in the directed edge representation, the edge is still considered to be undirected, so that when determining whether an edge can be added to a proto-child, its sense may be reversed if necessary. Thus, GNX^u based on undirected edges is different from the the same operator for directed edges.

To develop BMM^u, we must investigate minimal mutations for undirected edges. It is easy to see that the minimal mutation for this representation is the reversal of a sub-tour. (Although this operator is often referred to in the TSP literature as *inversion,* e.g. Whitley *et al.*, 1989, this term will be avoided here because of possible confusion with the more general re-linking operator; Holland, 1975.) For example,

$$\{14, 34, 35, 25, 26, 16\}^u = 143526^\pi \overset{\mathrm{BMM}^u}{\longmapsto} 125346^\pi = \{12, 34, 35, 25, 46, 16\}^u, \quad (7.12)$$

reversing the section from positions 2 to 5 inclusive. Minimal mutations are thus at distance 2 from their parents in this representation (sub-tour reversal breaks two edges), allowing us to implement BMM^u directly.

7.2.3 The Directed Edge Representation

The directed edge representation builds from the undirected edge representation introduced in section 7.2.2 by defining genes while still capturing the obvious structure presented by the edges contained in a tour. We employ a characterisation χ_d that generates basic equivalence relations which group together tours based on the city visited directly after a specified city. This leads to a (formal) genetic representation, where a tour is represented by the set of directed edges it contains. It is genetic because genes are well defined, the ith corresponding to the city visited after city i. Thus, identifying this representation by the superscript d,

$$1423^\pi = \{\vec{14}, \vec{23}, \vec{31}, \vec{42}\}^d, \quad (7.13)$$

or, stressing the genetic nature of the representation,

$$1423^\pi = \{(1,4), (2,3), (3,1), (4,2)\}^d \equiv 4312^d. \quad (7.14)$$

As with the permutation representation, the degeneracy arising from the freedom to choose the direction in which to traverse the tour may be formally removed by convention, but again this is not wholly satisfactory. This is discussed in section 7.2.6.

χ_d generates formae consisting of tours containing a specified set of (directed) edges. For example, the forma ξ' containing tours with the edge $\vec{21}$ is

$$\xi' = \{1234^\pi, 1243^\pi\} = \{4123^d, 3142^d\} \equiv \square 1 \square \square^d, \quad (7.15)$$

using the concise notation to indicate the value of each gene, and is described by

$$\langle \xi' \rangle = \{\vec{21}\}. \tag{7.16}$$

We can calculate the cardinality of arbitrary formae in this representation using the concept of threads introduced with the undirected edge representation. However, as we must now preserve the direction of the edges, we see that

$$|\xi| = (t(\xi) + t_0(\xi) - 1)! \tag{7.17}$$

with a divisor of two only introduced when $t(\xi) = o(\xi) = 0$. Again, we confirm that $|\xi'| = (1 + 2 - 1)! = 2$.

The construction of RAR^d and GNX^d for this representation is again straightforward. The minimal mutations for the directed edge representation are 3-changes, for example,

$$143526^\pi \overset{\mathrm{BMM}^d}{\longmapsto} 142356^\pi, \tag{7.18}$$

which corresponds to cycling the edges $\{\vec{43}, \vec{52}, \vec{26}\}$ to become $\{\vec{42}, \vec{56}, \vec{23}\}$. Minimal mutations are at distance 3 in this representation. The reader will quickly become convinced that such 3-changes suffice, in sequence, to generate all tours, and that 1-changes and 2-changes are not possible in this representation.

Attempting to construct an instance of the forma described by $\{\vec{23}, \vec{31}\}$ from parents 123456^π and 126543^π should convince the reader that transmission is incompatible with assortment here also.

7.2.4 The Corner Representation

The final representation is in some ways the most complex, and is again based on undirected edges, but forms a (formal) genetic representation, denoted c. In this case, we choose a characterisation χ_c that formalises a belief that solutions sharing both links to a city will tend to exhibit similar cost. This generates an equivalence relation (gene) corresponding to each city, each of which takes compound alleles consisting of the unordered pair of edges centred on that city. For example,

$$1423^\pi = \Big\{(1, \{3, 4\}), (2, \{3, 4\}), (3, \{1, 2\}), (4, \{1, 2\})\Big\}^c \tag{7.19}$$

where a tuple $(a, \{b, c\})$ indicates that city a has neighbours b and c.

Because genes in this representation define two edges, there are only trivial examples for $n = 4$. For example, the forma ξ' with edges to city 2 and city 4 from city 1 is

$$\xi' = \{1234^\pi\} \tag{7.20}$$

and is described by

$$\langle \xi' \rangle = \{(1, \{2, 4\})\}. \tag{7.21}$$

The cardinality of corner formae is calculated using exactly the same formula as was defined for undirected edges in section 7.2.2, e.g. since $t(\xi') = 1$, $t_0(\xi') = 1$, we confirm that $|\xi'| = (1 + 1 - 1)!\, 2^0 = 1$.

Defining the operators RAR^c and GNX^c is straightforward. It is easy to confirm that the minimal mutation in this representation is sub-tour reversal (a 2-change in terms of edges), so the same example as for undirected edges may be used, specifically

$$143526^{\pi} \overset{\mathrm{BMM}^c}{\longmapsto} 125346^{\pi}. \tag{7.22}$$

Notice, however, that in this case the new solution is at distance 4 from its parent in this representation, because genes 1, 2, 4 and 6 have changed.

The incompatibility of transmission and assortment may be verified for this representation by attempting to generate an instance of the forma described by

$$\{(4, \{3, 5\}), (6, \{3, 7\})\} \tag{7.23}$$

from parents 12345678^{π} and 12367458^{π}.

7.2.5 Linkage Considerations

The positioning of genes on a chromosome defines their *linkage*. With crossover operators based on transferring contiguous portions of the genome from parents to children, such as N-point crossover, linkage can significantly affect algorithmic performance. In the case of the permutation representation, where the ith gene specifies the ith city visited, the static linkage achieved by placing the ith gene at the ith locus seems natural. With the other genetic representations—directed edges and corners—and the pseudo-genetic representation based on undirected edges, it is less clear that placing the ith gene at the ith locus is sensible. This is because in each of these cases, the ith gene is associated with the ith-*labelled* city, rather than the ith city in the tour, and the city labels are (usually) arbitrary. A simple way to determine the linkage adaptively is to re-link the genome in one of the two possible orders achieved by following the tour in a consistent direction. It seems at least possible that GNX using this linkage will perform better than using the essentially random linkage achieved by determining locus from city number. Both forms of linkage will therefore be tried with GNX.

7.2.6 Redundancy and Degeneracy

All four representations considered contain some redundancy, because the last allele value can always be determined from the others (owing to the cyclic nature of the tour). The corner representation, however, contains vastly more than the other representations, actually specifying each edge twice. Thus fully half of the genetic material in the corner representation is formally redundant—specifying every other corner in the

tour would suffice. Notice, however, that no *fixed* set of corners (those centred on a particular half of the cities) would be adequate, which is why the representation used specifies the corner for each city.

Both the permutation representation and the directed edge representation exhibit degeneracy, while the undirected edge representation and the corner representation do not. While folding out the factor of n arising from the need to fix a starting city is easily handled, the degeneracy associated with the direction of travel is more problematical.

There are a number of possible responses to degeneracy:

1. *Gauge fixing.* As has been stated earlier, the problem can be removed formally by a convention such as that used in this chapter—always choosing the direction so that the second city has a lower numeric label than the nth. The problem with this is that some very similar solutions have almost maximally different representations. For example, 162453^{π} is a minimal mutation of 126453^{π}, but in standard form is represented as 135426^{π}. This is a practical problem for recombination, which will fail to recognise that the two solutions have anything in common except the redundant first gene and "equidistant" fourth gene, and leads to a certain "brittleness".

2. *Ignore the problem.* It is possible simply to leave the evolutionary search to use two different representatives for each solution. This avoids the brittleness of the "gauge fixing" approach, but at the cost of searching a larger space with two optima, and (more importantly) failing to allow recombination to recognise even when it is manipulating identical parent solutions. While there is some evidence that evolutionary search is still reasonably effective in these circumstances, it is hard to avoid the suspicion that its efficiency is reduced.

3. *Align before recombination.* The practical difficulties of degeneracy manifest themselves during recombination. An alternative approach involves computing the distance between the first parent the second traversed in one sense, and then computing the distance with the second parent reversed. Recombination is performed with second parent in the sense that minimises the distance. This is similar in spirit to an approach used by Montana & Davis (1989) for removing the well-known hidden node degeneracy in feed-forward neural networks before recombination. This avoids the problem of brittleness and recombination's consequent inability to recognise certain very similar solutions, but is arguably somewhat arbitrary.

All of these options have drawbacks. For this study, the second option has been chosen as the simplest, though the third option also has a certain attraction.

7.2.7 Characteristics of TSP Representations

Before proceeding to the experimental results, it will prove useful to summarise the key characteristics of the four representations considered. These are presented in table 7.1.

representation	rep. type	cardinality	redundancy	degeneracy	MM-order (edges)	MM-order (genes)	linkage	variance
permutation	genetic	$O(n)$	low	$\times 2$	4	2	good	high
directed edge	genetic	$O(n)$	low	$\times 2$	3	3	random	medium
undirected edge	allelic	—	low	none	2	2	none	medium
undirected edge	pseudo	$O(n)$	low	$\times 2$	2	2	random	medium
corner	genetic	$O(n^2)$	high	none	2	4	random	low

Table 7.1: This table shows the main characteristics of the representations studied for the TSP. Cardinality indicates the number of allele values per gene (where applicable). Degeneracy and redundancy are discussed in section 7.2.6. Mutation order lists first the number of edges and second the number of gene values changed by a minimal mutation. The linkage column describes the appropriateness of the "natural" linkage for the representation, and variance gives the relative fitness variance of formae, as shown in figure 7.2. Note that the undirected edge representation is shown twice, once in its allelic form and once as the pseudo-genetic representation discussed in section 7.2.2.

7.3 Forma Variance Measurements

Motivated by the discussions in chapter 6, we examine the relationship between forma variance and forma order and size for the four representations introduced above.

Figure 3.7 (page 43) shows the standard deviation of fitness (cost) within formae as a function of forma order for the four different representations. Each point is based on 100 samples from each of 100 formae of the given order, for the 100-city Krolak 'C' problem from TSPLIB (Reinelt, 1990). Graphs of higher fitness moments are qualitatively rather similar, and are not shown.

In order to account for the varying levels of redundancy and degeneracy, an arguably fairer comparison is achieved by presenting the results in terms of forma size. This is shown in figure 7.2, where we see that the corner and undirected-edge representations appear to be best, followed by the directed edge representation (which progressively degrades for smaller formae), and finally the permutation representation. In the following section we will show that this ordering is an excellent indicator of the performance of algorithms based on these representations.

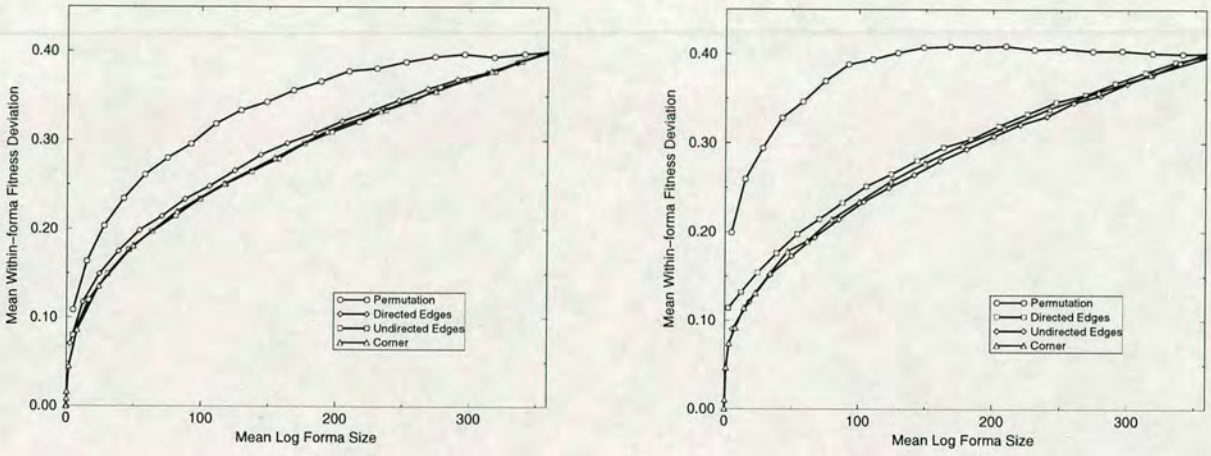


Figure 7.2: The graphs show the mean standard deviation of fitness (cost) within randomly generated formae (left), and randomly generated formae containing the optimal tour (right), plotted against mean log forma size for each of the four representations considered in this chapter. Each plotted point is based on 200 sampled tours drawn at random from each of 200 formae of fixed order, for the Krolak 'C' problem. Confidence intervals (standard errors) are omitted as they are smaller than the plotted symbols. Plotting against forma size arguably gives a fairer comparison than plotting against order directly (compare 3.7). Note that all edge-based representations show significantly lower variance than the permutation representation, and we can also see the benefit of removing the artificial directional constraints in the two undirected representations. These effects are particularly emphasised when "good" formae (containing the optimum) are considered—in fact for the permutation representation the fitness deviation actually rises as formae size decreases! (Note however that only the left graph can reasonably be regarded as a diagnostic tool as it can be generated before the optimum is known.)

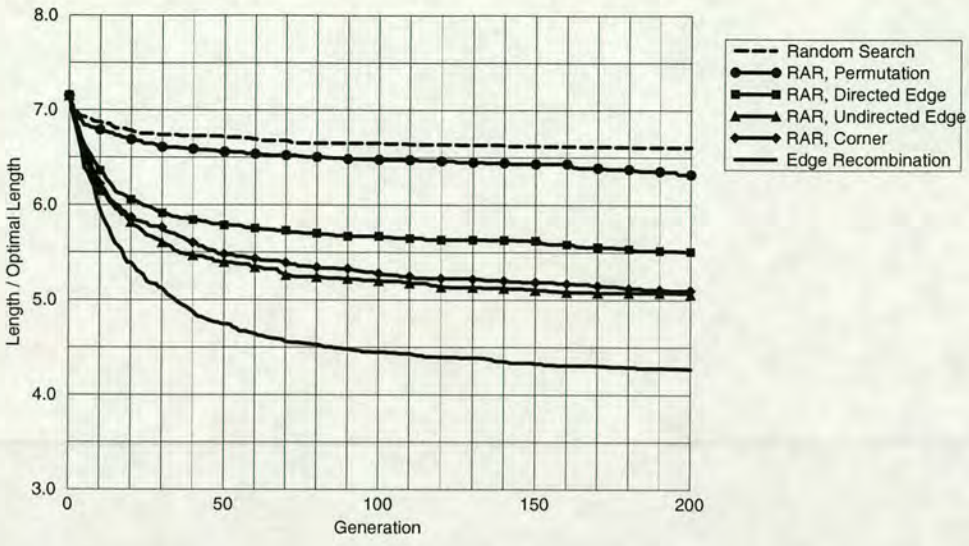


Figure 7.3: The graph shows the results produced by representation-independent genetic algorithms on the 100-city Krolak 'C' problem, instantiated with four representations using the RAR recombination operator. For comparison, a curve for random search is shown, as is Whitley *et al.*'s edge recombination operator, modified by deferring patching of tour fragments until all parent edges are exhausted.

7.4 Experimental Results

TSP optimisation experiments were then conducted using genetic algorithms, using RAR_2 and G2X for recombination, as well as edge recombination[1] for a domain-specific comparison. For all representations except permutations, G2X was applied with both fixed linkage and the "tour-following" scheme described in section 7.2.5. The order of application of operators was recombination (with probability 1.0) of two parents chosen by probabilistic binary tournament selection, followed by BMM (defined with respect to the chosen representation) with a rate chosen so that the mean number of mutations per chromosome was one. A panmictic population of size of 100 was used with a steady state update scheme. All performance results are averages over 20 runs, and show the length of the best tour in the population relative to the optimum. Each representation started from the same set of 20 randomly-generated populations. Error bars are omitted since they are in all cases smaller than the tick sizes.

Both the 100-city Krolak 'C' and the 442-hole PCB drilling problem from TSPLIB were studied. The genetic algorithm for the smaller problem used probabilistic binary tournaments with $p = 0.7$ for selection and replacement, with elitism, and allowed duplicates. Results for these runs are shown in figures 7.3, 7.4 and 7.5.

The larger problem used a more aggressive GENITOR-style plan adapted from

[1]The edge recombination operator, defined in Whitley *et al.* (1989), seeks to maximize edge transmission by overlaying the two parent tours to form an edge map, and then iteratively adding outlying edges from this map to the proto-child. In contrast, RAR_2 iteratively adds random edges from the parent tours to the proto-child.

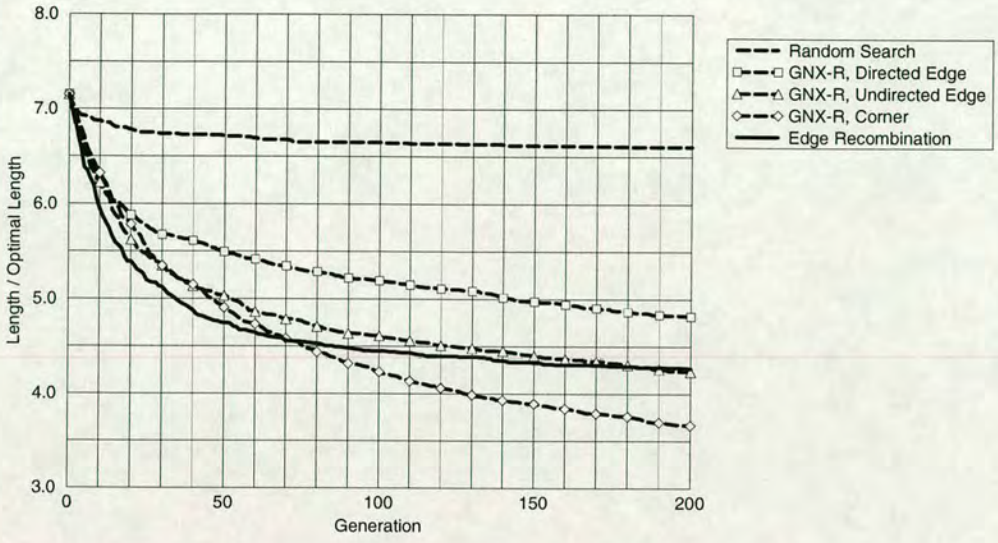


Figure 7.4: The graph shows the results produced by representation-independent genetic algorithms on the 100-city Krolak 'C' problem, using the GNX-R recombination operator (c.f. figure 7.3). These runs use the "natural" linkage associated with the representation. For the edge-based and corner representations, this linkage is essentially random (the gene's locus being determined by its city label).

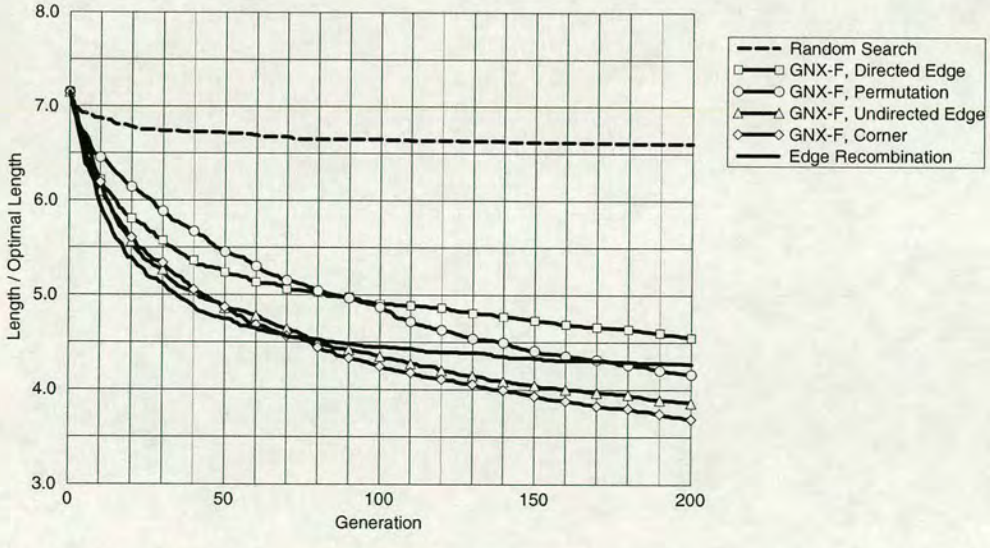


Figure 7.5: The graph shows the results produced by representation-independent genetic algorithms on the 100-city Krolak 'C' problem, using the GNX-F recombination operator (c.f. figures 7.3, 7.4). These runs use a dynamic "tour-following" linkage.

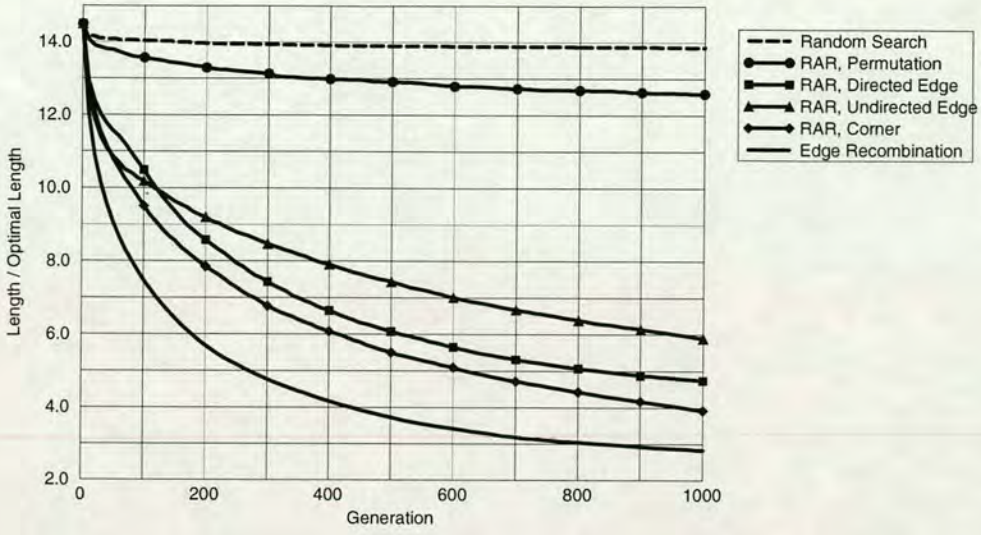


Figure 7.6: The graph shows results for a GENITOR-style genetic algorithm with RAR on the 442-hole PCB drilling problem corresponding to those in figure 7.3.

Whitley *et al.* (1989), using tournament selection with probability 1.0, and replacement of the worst individual. Duplicate solutions were forbidden. For edge recombination only, a zero mutation rate was used in line with its creators' recommendation (to achieve maximal edge transmission). Results for the genetic algorithms are shown in figures 7.6 and 7.7 (also compare figure 9.5).

In general, the results show fitness variance of formae to be a powerful indicator of algorithmic performance. (For simplicity we consider only the natural "best so far" performance metric.) The results expected on the basis of figure 7.2 would be that for any fixed algorithm, corners and undirected edges should perform best, permutations worst, with directed edges somewhere in between. The only discrepancies that need to be explained are now addressed in turn.

First, although figure 3.7 suggests that undirected edges should exhibit performance similar to directed edges, we find that in four of the five direct comparisons, undirected edges out-perform directed edges despite similar forma variance as a function of forma order. Factors explaining this might include the disparity in size of directed- and undirected-edge forma of the same order, the difference in mutation operators (two-changes for undirected edges versus three-changes for directed edges) and the greater disruptiveness of recombination for directed edges. The one case in which directed edges do better is a very aggressive plan with RAR. Here, it is possible that mutation is performing a more important search rôle, and greater recombinative disruption effectively increases the mutation rate advantageously.

Secondly, while permutations generally perform poorly, as forma variance would suggest, in one case (GNX on the 100 city problem) they perform rather better than

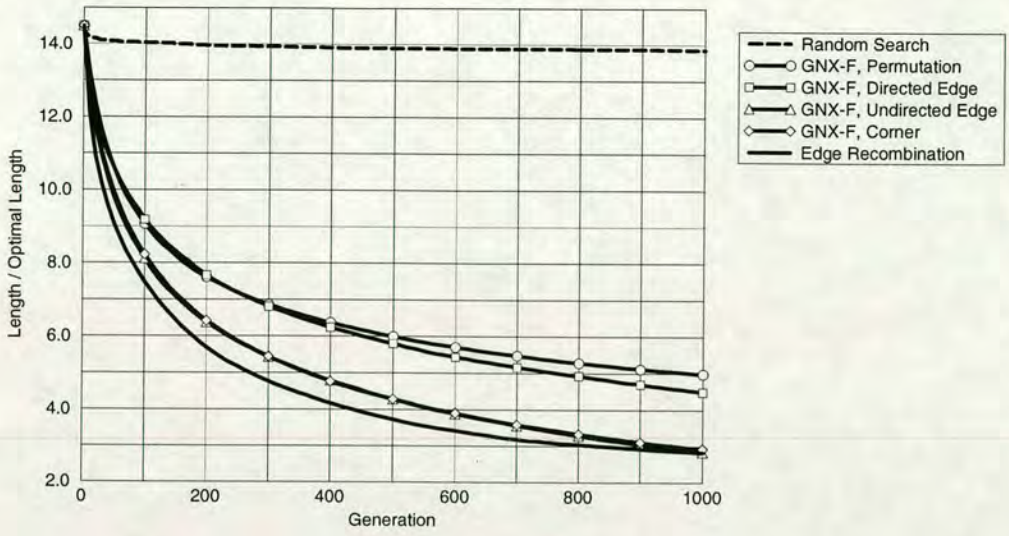


Figure 7.7: The graph shows results for a GENITOR-style genetic algorithm with GNX on the 442-hole PCB drilling problem corresponding to those in figure 7.5. Results for the inferior random linkage corresponding to figure 7.4 are omitted.

might be expected. This is probably in part because GNX takes long contiguous chunks from parents, thus effectively transmitting many edges even for permutations. (That is, for small values of n, GNX_π is effectively "processing" edges to some extent.) As noted in figure 7.2, if the formae considered are restricted to be those with contiguous defining positions, forma variance falls almost exactly to that for the edge-based representations, largely explaining this anomaly.

The final discrepancy concerns the performance of the corner representation relative to that of undirected edges. Here, the general pattern is that when good linkage is used and maintained, performance is very similar, but when poorer linkage is used, or is severely disrupted (as with RAR), corners significantly outperform edges. This is wholly understandable since corners intrinsically carry much more linkage information, in the sense that every corner specifies an adjacency of two edges. The only case in which this pattern breaks down is for the smaller problem where corners and undirected edges perform similarly with RAR.

Other points evident from the results are that GNX consistently out-performs RAR and that linkage effects are (perhaps unsurprisingly) rather strong. More surprisingly, GNX with both the corner and undirected edge representations, even with non-optimised parameters, appears at least competitive with, and arguably superior to, edge recombination. It is encouraging to find that although GNX is a completely generic (domain-independent) recombination operator, when *instantiated* using an appropriate representation it is competitive with a domain-specific operator developed explicitly for the TSP.

7.5 Discussion

The general pattern of results supports the hypothesis that the fitness variance of forma exhibited by a representation acts as a good predictor of its performance in formal genetic algorithms. Indeed, given the large number of potentially relevant differences between the four representations considered, the predictive power of forma variance is rather impressive. While results have only been gathered for one problem class (the travelling sales-rep problem) and a limited range of representation-independent algorithms, they provide a powerful case for corresponding studies in other problem domains.

The best results overall were produced with the corner representation, which has a number of unusual features. Principal among these is its extremely high cardinality and compound allele structure. The results therefore provide further evidence that the traditional advocacy of low cardinality representations as universally appropriate is misguided. (It might prove instructive to investigate even higher-cardinality "corner" representations—specifying three or more edges per gene—in order to discover when performance begins to deteriorate.)

The results obtained with GNX are at least competitive with, and arguably superior to, those obtained with edge recombination. Since this is widely regarded as one of the best forms of recombination for the TSP when a genetic algorithm without local search is used (e.g. Freisleben & Merz, 1996), this is a significant finding. Moreover, since GNX is applicable to *any* (formal) genetic representation (including non-orthogonal representations) it may well prove effective in other problem domains. These studies have demonstrated that the construction of formal representation-independent operators and algorithms is not merely of theoretical import, but can provide competitive practical search tools.

Linkage has long been recognised as an important theoretical characteristic of chromosomes in the context of recombination, but has rarely been shown to have a major effect on performance in practice. These experiments have clearly demonstrated linkage effects, and shown that adaptive linkage strategies—albeit not the traditional inversion-based approach—can yield superior performance. The best results achieved were with corners, which contain intrinsic linkage information, and undirected edges when linked by tour following.

Chapter 8

Case Study: Real-parameter Optimisation

We are convinced by things that show internal complexity, that show the traces of an interesting evolution. These signs tell us that we might be rewarded if we accord it our trust. (Brian Eno)

Many optimisation problems are formulated as a search for vectors of real-valued parameters that form extrema of some function. A variety of both local and global techniques with varying degrees of specialisation have been proposed for tackling such problems. Evolutionary algorithms have also been regularly applied in these domains, a particular attraction being that they require only the ability to evaluate the function at any point. Indeed, evolution strategies have been primarily focused on real-parameter optimisation, with theoretical results specialised to this domain. Traditional genetic algorithms, on the the other hand, are applied to such problems by mapping to a canonical representation space of binary strings for which simple operators are defined. Typical "practical" genetic algorithms specialise these operators by considering the phenotypic effects of the moves they generate in the search domain of real parameters.

In this chapter, we demonstrate deep connections between the approaches favoured for continuous domains in evolution strategies and "pragmatic" genetic algorithms, and the operators developed in the "traditional binary" genetic algorithm school for combinatorial optimisation. This is achieved through exploiting the formal procedure for transferring algorithms and operators between arbitrary search domains developed in chapters 3–6.

We show that by explicitly designing representations that capture beliefs about the structure of the search domain of real parameters (such as the importance of locality and continuity), we can instantiate problem-independent algorithms built from generalised mutation and recombination operators (exactly as we did in a combinatorial optimisation domain in chapter 7). We find that when particular characterisations are employed, we derive commonly used operators such as blend crossover, line recombination and gaussian mutation from the generalised operators.

To facilitate this, we extend previous work on formal construction of representations in discrete (typically combinatorial) search problems to continuous domains. A sequence of representations forming increasingly accurate approximations to the continuous space is employed, and requirements for the limiting process are formulated. When conventional representations for real parameters are considered within this framework, previously suspected peculiarities in their behaviour are confirmed.

We proceed to develop two formal representations for real-parameter evolutionary optimisation based on a formal codification of beliefs about the nature and structure of continuous search domains. We have named the resulting representations the *Dedekind* and *Isodedekind* representations, for reasons that are explained later. We examine the limiting behaviour of these representations, and derive problem-specific forms of generic genetic operators introduced previously. These are seen to reduce serendipitously to 'sensible' operators already widely used for practical optimisation.

Having constructed the new representations, we find ourselves able to apply identical (formal) genetic move operators, and therefore identical formal evolutionary *algorithms* with four different representations of real parameter spaces—"traditional" binary coding, Gray coding, Dedekind and Isodedekind. We observe striking qualitative differences in behaviour of these four representations for even the simplest objective functions (figure 8.1).

The primary purposes of this work are to illuminate deep connections between "evolution strategy style" and "genetic algorithm style" operators, to bridge a gap between discrete and continuous domains, and to expose the formal gene structure underpinning evolutionary approaches to continuous optimisation. It also, however, provides a study in the formal construction of representations and operators from explicit codifications of beliefs about the structure of search domains. In this connection, this work also demonstrates convincingly that the characterisation of a particular problem domain used to induce a representation need not be completely free of 'conflicting' beliefs. Although such characterisations can lead to highly *non-orthogonal* representations (in which the legal values for a given allele are dependent on the current values of others), this is not seen to be problematic in general—indeed the operators derived from such representations may be more powerful than those resulting from simpler ones.

8.1 Evolutionary Real-parameter Optimisation

Optimisation of functions defined over real parameters has a long history, so it is natural that the area has received significant attention from the evolutionary algorithms community. Several now converging schools of thought have brought different points of view to the attack.

In the evolution-strategy paradigm (Rechenberg, 1973) and in the evolutionary-

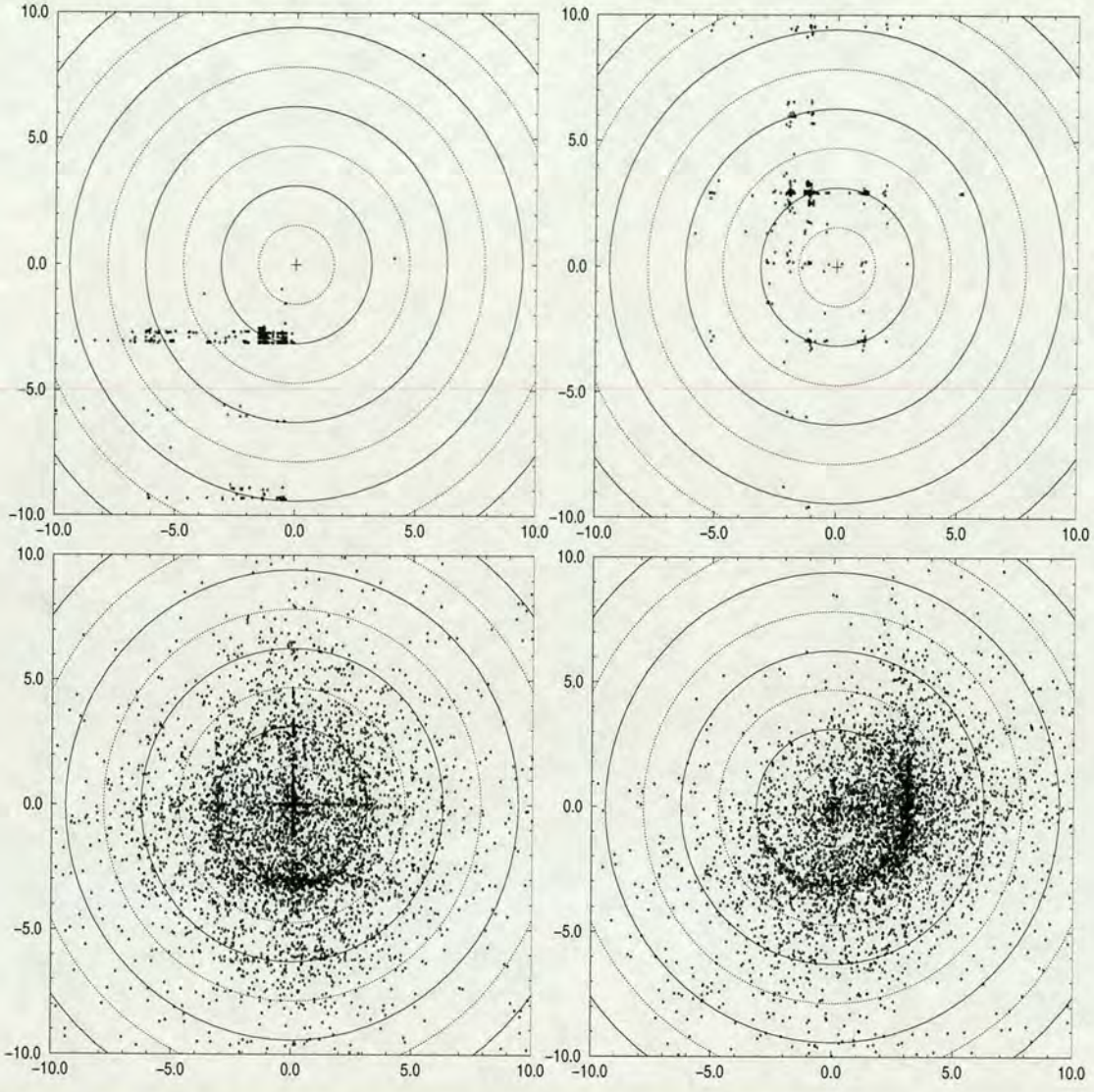


Figure 8.1: In the spirit of the work presented in Eshelman & Schaffer (1992), this figure illustrates the results of applying an identical formal algorithm instantiated with the four different real-parameter representations discussed herein to Schaffer's F6 function. (Eshelman & Schaffer, 1992). The two-dimensional function is radially symmetric with global maximum at $(0, 0)$, local maxima on circles of radius $\pi, 2\pi, \ldots$, and local minima on circles of radius $\pi/2, 3\pi/2, \ldots$. The search domain is $[-100, 100] \times [-100, 100]$ of which the figures show the central region. Each figure shows all of the points sampled in one typical run of a fixed algorithm based on the R^3 and BMM operators (see chapter 5). Each algorithm sampled approximately 9000 points in the central region during a run of 100 generations with population size 100. The binary representation (top-left) exhibits extremely poor coverage and an extremely striated sampling pattern based on the relative periodicity in the function and the representation. A Gray-coded representation (top-right) typically shows better coverage, as mutation is more effective, but the sampling pattern is clearly biased. The Dedekind representation (bottom-left) shows much better coverage, but since R^3 reduces to BLX-0 (see figure 8.2) there is still a tendency to favour the axis directions, and an inward bias on the population. The algorithm based on the Isodedekind representation (bottom-right) still shows the inward population bias since R^3 reduces to line recombination, but the axial skew is removed.

programming school (Fogel *et al.*, 1966), the vector of parameters is typically interpreted directly as a "genome", with "gene" values approximated by floating-point machine values. A variety of specialised recombination and mutation operators that directly manipulate these parameters have been employed, but to date their connection to operators used in other search domains has not been apparent. Work with evolution strategies (Bäck & Schwefel, 1993) stresses the importance of gaussian (creep) mutation (possibly with adapted width) as a search operator, based on the so-called *principle of strong causality,* by which small changes in parameter values are assumed to lead to small changes in the objective function. We will see later that by formalising this or other beliefs about the search domain, we can gain insight into what structures the resulting algorithms may be said to be "processing". Recombination methods including line recombination ($\boldsymbol{z} = \alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}$), parameter-wise uniform crossover (also known as local-discrete recombination) and blend crossover (local-intermediate recombination) have been employed.

In the genetic-algorithm school, this "parameters-as-genes" approach has not been universally accepted. The common practice has been to represent and manipulate real parameters as fixed-length binary strings using either "traditional" integer coding (where bit strings are decoded as integers and linearly scaled to the appropriate parameter ranges) or "Gray coding" (in which consecutive integers are coded by bit strings that differ in only a single position). Such binary codings allow "standard" genetic operators such as N-point crossover and point mutation to be applied in real-parameter optimisation, albeit with the restriction that the discretisation must result in 2^k points per parameter, for some integer k. They also reflect continuing attachment by many to the dubious *principle of minimal alphabets* (Goldberg, 1989c), which has been shown to be motivated by highly questionable theoretical observations (Radcliffe, 1991a; Vose & Liepins, 1991b).

An increasing proportion of the genetic algorithms community, however, particularly those working on real-world applications, have pointed out the efficacy of working directly with the real parameters (e.g. Davis, 1991b; Michalewicz, 1992). Using "standard" genetic operators in this case—viewing parameters as genes—is problematical as has been pointed out by Goldberg (1990), with the result that *ad hoc* operators have been generally been used, such as "creep mutation" (Davis, 1991b) and blend crossover (Eshelman & Schaffer, 1992; generalised from the R^3 operator of Radcliffe, 1991a). Although practically useful, such approaches have lacked a formal basis (for instance, it is not clear what a gene is or how the genetic operators are formally defined), and operators are seen conceptually as acting directly in the search space rather than in a space of genotypes that represents it. In the coming sections we will show that both of the standard evolution-strategy style operators for reals and the standard genetic

Evolution strategy term	Genetic algorithm term	Formal derivation
Local-discrete recombination	Uniform crossover	RAR, RTR + Real
Local-intermediate recombination	BLX-0	RAR, R^3, RTR + Dedekind
Line recombination	Line recombination	RAR, R^3, RTR + Isodedekind
n/a	N-point crossover	$\sim$ GNX + Real
Gaussian mutation	creep mutation	BMM + Dedekind or Isodedekind
n/a	parameter-wise mutation	BMM + Real
n/a	bit-wise point mutation	n/a

Table 8.1: The table summarises several of the operators commonly used in genetic algorithms and evolution strategies (often with different names). Although they may appear to be completely different from one another, they can all be derived as problem-specific forms of generalised problem-independent operators when particular representations are chosen. The "formal" operators RAR, RTR, R^3, GNX and BMM are described in chapter 5, BLX-0 is blend crossover as illustrated in figure 8.2, and the Dedekind and Isodedekind representations are described in sections 8.3.3 and 8.3.4 respectively.

algorithm operators can be derived from common "representation-independent" template operators. The resulting equivalences are shown in table 8.1. Figure 8.2 shows graphically the (phenotypic) effect of the different recombination operators, and figure 8.3 illustrates the various mutation operators.

In addition to the need for satisfactory mathematical operator derivations, in the case of continuous domains there is a need for clearer understanding of the relationship between operators' effect in a discrete approximation space and in the underlying continuous space. In particular, it seems desirable that operators have well-defined behaviour as the grid spacing shrinks to zero, or at least that we understand what is happening if this is not the case. While gray coding has traditionally been put forward as a "smoother" binary representation for reals, the analysis here will show that it shares with "traditional" binary coding pathological limiting behaviour, while the Dedekind and Isodedekind representations have well-behaved natural continuous limits.

8.2 Scaling Properties of Discretised Representations

Although previous work on forma analysis has concentrated primarily on finite search domains, such as combinatorial problems like the travelling sales-rep problem (Radcliffe & Surry, 1994b), neural network topology optimisation (Radcliffe, 1993), multi-objective pipeline optimisation (Surry *et al.*, 1995) and so forth, some initial work was done on continuous domains (Radcliffe, 1991a, 1991b). In this chapter we extend these ideas by considering a limiting sequence of discrete representations. These results are used to define two formal genetic representations for real-parameter optimisation, which

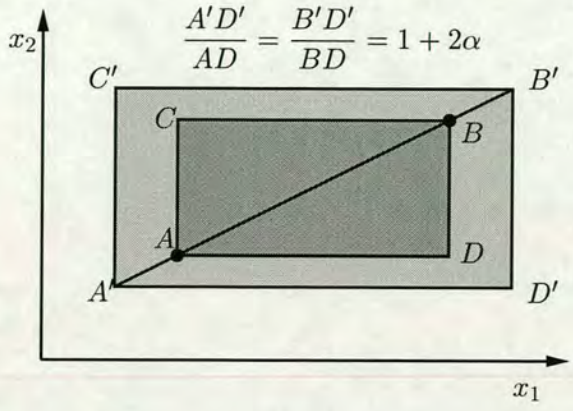


Figure 8.2: The figure illustrates some of the variety of crossover operators typically used in evolutionary optimisation of real parameters. Here, parents A and B, each a two-component vector, are crossed. Parameter-wise uniform crossover (termed discrete recombination in evolution strategies) generates A, B, C or D with equal probability. For the formal real representation, RAR, RTR, and R^3 are all equivalent to blend crossover with parameter 0 (BLX-0; termed intermediate recombination in evolution strategies), and generate a child uniformly from the rectangle $ACBD$. BLX-α ($\alpha > 0$) generates a child uniformly from the rectangle $A'C'B'D'$. Line recombination generates children uniformly on the line AB, and extended line recombination generates children uniformly on the line $A'B'$. Standard (N-point and uniform) operators with traditional binary codings generate non-localised children that are difficult to show schematically.

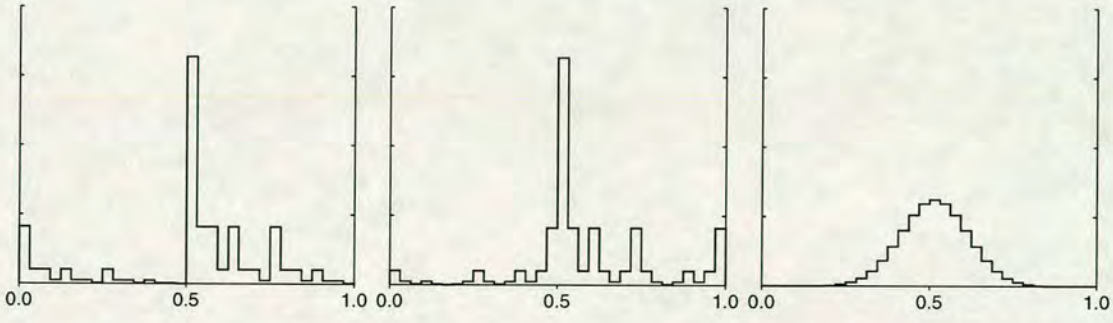


Figure 8.3: The figure illustrates the effects of instantiating the representation-independent mutation operator BMM (see chapter 5) for traditional binary coding (left), Gray coding (centre) and the Dedekind representation (right). For the first two representations, BMM reduces to standard point mutation, and in the last case to gaussian creep mutation. For each representation, the real interval $[0, 1]$ has been discretised into 32 points, and the graphs show the probability distribution over possible offspring when the genome representing the point just right of 0.5 is mutated. The pathological distributions in the first two cases help explain some of the behaviour observed in figure 8.1, where BMM is employed (as point mutation) for the two binary representations.

are used to derive problem-specific forms of the generalised genetic operators.

The methodology of chapter 4 has focused on the representation of discrete spaces. However, we seek to extend our formalism to the case of continuous spaces such as the reals. It is clear that any implementation of a search algorithm using digital computers will be finite, so that we are forced to consider an *approximation* of some kind to the actual search space. It is the goal of this section to formalise the requirements that we might enforce in order to make this approximation meaningful.

First of all, we require that we can (in principle) generate a representation of the continuous search space to any desired level of accuracy. We then require that the actions of our search operators have sensible limiting behaviour as we arbitrarily increase the accuracy of our approximate search space.

Consider a problem domain $\mathcal{D}$ in which each problem instance I is defined on a continuous search space $\mathcal{S}$, typically a subset of $\mathbb{R}^m$. Suppose further that there is a metric, $d : \mathcal{S} \times \mathcal{S} \longrightarrow \mathbb{R}^+$ associated with $\mathcal{S}$. We wish to generate an approximate representation for any given problem instance in the domain, to any degree of accuracy. Let $n \in \mathbb{Z}^+$ indicate the degree of accuracy desired, as formalised below. We suppose that a characterisation, $\chi(I, n)$, is available, which is an automatic procedure for generating a finite representation space $\mathcal{C}_{\chi(I,n)}$ and growth function $g_{\chi(I,n)}$ (which we will abbreviate as $\mathcal{C}_n$ and g_n). The growth function maps chromosomes in the representation space into structures into a finite subset $\mathcal{S}_n$ of the search space, as illustrated in figure 8.4.

We first require that the approximation of $\mathcal{S}$ can be made arbitrarily good. Formally, we require that for any open subset of $\mathcal{S}$, there is some level of accuracy above which our approximation always represents some point in any given subset:

$$\forall \mathcal{B} \subseteq \mathcal{S} \ (\mathcal{B} \text{ open}) \ \exists n_0 \in \mathbb{Z}^+ \ : \ n > n_0 \Longrightarrow \mathcal{S}_n \cap \mathcal{B} \neq \varnothing, \tag{8.1}$$

where $\mathcal{S}_n = g_n(\mathcal{C}_n)$ is the subset of the search space currently represented.

Furthermore, we require that any search operators to be used exhibit reasonable limiting behaviour. Thus, as we change the degree of accuracy of our approximation, we desire that the action of the operators in the search space does not change radically with respect to the distance function defined on $\mathcal{S}$.

As in chapter 5, any probabilistic search operator, ω, can be viewed as generating a child chromosome from one or more parent chromosomes

$$\tilde{\omega}(x_1, \ldots, x_q) \triangleq \omega(x_1, \ldots, x_q, \kappa) \ : \ \kappa \sim U(\mathcal{K}_\omega). \tag{8.2}$$

where $\tilde{\omega}$ is based on the deterministic operator ω with its control parameter selected uniformly from a control set.

In order to restrict the limiting behaviour of such an operator, we require that its action on chromosomes representing nearly the same point in the search space tends to

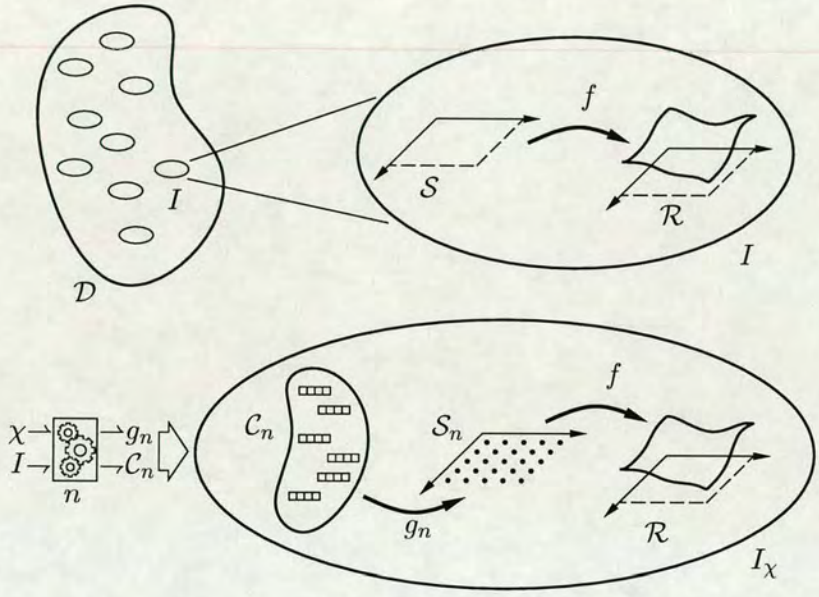


Figure 8.4: A *problem domain* $\mathcal{D}$ consists of a set of *problem instances*. Each instance I defines a *search space* $\mathcal{S}$ of candidate solutions, a *fitness function* f and a set of objective values $\mathcal{R}$. A *characterisation* χ of the domain specifies a set of equivalences among the solutions for any instance I, for any given accuracy n. These equivalences induce a *representation* made up of a *representation space* $\mathcal{C}_{\chi(I,n)}$ (of *chromosomes*) and a *growth function* $g_{\chi(I,n)}$ mapping chromosomes to a subset $\mathcal{S}_n$ of $\mathcal{S}$. A chromosome x is a string of *alleles*, each of which indicates that x satisfies a particular equivalence on $\mathcal{S}$. Algorithms can be completely specified by their action on the alleles of these generalised chromosomes, making them totally independent of the problem domain itself.

be the same. Formally, for a q-ary operator, we require that

$$\forall \varepsilon > 0\ \forall \{(a_i, x_i)\}_{i=1}^{q}\ (a_i \in \mathcal{S}, x_i \in \mathcal{C})\ \forall \mathcal{B} \subseteq \mathcal{S}\ (\mathcal{B} \text{ open}) \exists \delta, p_0 > 0\ \exists n_0 \in \mathbb{Z}^+ :$$
$$n > n_0, \max_{1 \le i \le q} d(g_n(x_i), a_i) < \delta \Longrightarrow |P(g_n(\tilde{\omega}(x_1, x_2, \ldots, x_q)) \in \mathcal{B}) - p_0| < \varepsilon. \quad (8.3)$$

Thus the probability that the operator generates a representative of a point contained in any fixed open subset ($\mathcal{B}$) of $\mathcal{S}$ must converge to a fixed value (p_0) as the operands (the x_i) converge to representatives of any fixed set of points (the a_i) in $\mathcal{S}$.

8.3 Representations for Real-parameter Optimisation

In this section, we consider a number of different representations for discretised real-parameter optimisation. We first discuss traditional integer-coding and the related Gray coding, and then go on to develop two new representations and their associated operators based on formally characterising our beliefs about the structure of the search domain.

In generic real-parameter optimisation, an obvious belief we might like a representation to capture is some form of Lipschitz condition (also known as Hölder-continuity)—that small changes in the parameters lead to small changes in the observed function values (e.g. see Zhigljavsky, 1991). That is to say, we might believe that neighbouring solutions in the search space are expected to have related performance. By capturing this idea on an "axis-by-axis" (parameter by parameter) basis, we develop the Dedekind representation, and by considering all possible axis orientations simultaneously we derive the Isodedekind ("Isotropic Dedekind") representation. The operators we derive from these representations have been commonly used in evolution strategies, but are now shown to be formally equivalent to operators used in completely different discrete, combinatorial domains.

Traditional approaches to binary coding of real parameters is based (if on any explicit foundation!) on the belief that more schemata are better than fewer (the notion of implicit parallelism, giving rise to the principle of minimal alphabets). This has repeatedly been shown to be little more than statistical sleight of hand: one sample is one sample, not many. There is also perhaps some idea that because binary coding "chops up the search space" in many different ways, it lets the algorithm discover useful patterns (Goldberg, 1989c; Holland, 1975) but research has shown that is only true if the problem happens to coincide with the particular scaling and location captured in the binary coding. For example, Eshelman & Schaffer (1992) found that simply rescaling or shifting the coordinate axes could dramatically impact performance. (It is reasonable, however, to speculate about constructing a formal representation based on beliefs about periodicity in the objective function—or, indeed, on any other feature thought to be

relevant—but this has not as yet been achieved, although an early attempt was made by Radcliffe, 1991a.)

8.3.1 Approximating the Search Domain

In evolutionary real-parameter optimisation, we face the problem of representing a discrete approximation to a continuous search space. Here the search space is taken to be a product of real intervals,

$$\mathcal{S} \triangleq \prod_{i=1}^{m} [\alpha_i, \beta_i] \subseteq \mathbb{R}^m. \tag{8.4}$$

We consider a discretised approximation to the search space generated by the intersection points of a uniform lattice of planes along each co-ordinate axis, namely:

$$\mathcal{S}_n \triangleq \prod_{i=1}^{m} \{\alpha_i, \alpha_i + \Delta_{i,n}, \alpha_i + 2\Delta_{i,n}, \ldots, \beta_i\} \subset \mathcal{S} \tag{8.5}$$

where

$$\Delta_{i,n} = \frac{\beta_i - \alpha_i}{n-1}. \tag{8.6}$$

It is thus clear that by using a simple linear transformation, it is sufficient to consider the case $\mathcal{S}_n = \mathbb{Z}_n^m$. (For the traditional binary codings, we can, for simplicity, restrict n to powers of 2, but this is not a significant restriction.)

We begin by considering the case $m = 1$ (in which we represent a single parameter), and examine the traditional binary representations as well as developing the *Dedekind* representation. In section 8.3.4, we show that building up to higher dimensional search spaces is straightforward, and also introduce the alternative *Isodedekind* representation.

8.3.2 Traditional Genetic Algorithm Representations

We first examine the traditional methods for coercing a genetic algorithm into a real-parameter optimisation domain. The two standard and variously-championed approaches are traditional integer coding, in which an integer is directly coded in its base 2 representation, and Gray coding, which alleviates one of the perceived problems with the first approach (for instance, see Caruana & Schaffer, 1988). We present formal definitions of the representations in forma analysis terms, in order that we will later be able to derive forms of generalised genetic operators and to examine their limiting behaviour as we approximate the continuous space more closely.

We will see that neither representation is based on explicitly characterising any particular structure of the underlying optimisation problem, and that this leads to pathological limiting behaviour, not to mention poor performance on simple objective functions.

Traditional Integer Coding

A common approach to representing n adjacent integer values is to use $k = \lceil \log_2 n \rceil$ bits. When such an approach is taken, there is a choice, in principle, of $2^k!$ mappings between the 2^k values and the 2^k strings used to represent them (e.g. see figure 3.5). In practice, almost all such work uses either "traditional" binary coding, in which the ith value is represented by the binary number i, or so-called *Gray coding* (explained below).

With traditional integer-coding, the characterisation χ_{Integer} used to generate the representation is seen to consist of the k formal equivalence relations

$$\psi_i(x, y) = \begin{cases} 1, & \text{if } x \otimes 2^i = y \otimes 2^i, \\ 0, & \text{otherwise.} \end{cases} \tag{8.7}$$

where $i \in \{0, 1, \ldots, k-1\}$ and $\otimes$ denotes bitwise-and. These equivalence relations induce the basic formae (alleles) $\xi_0^0, \xi_0^1, \ldots, \xi_i^0, \xi_i^1, \ldots, \xi_{k-1}^0, \xi_{k-1}^1$ which we can identify exactly with schemata

$$\begin{aligned} \xi_i^0 &\equiv \square \cdots \square 0 \square \cdots \square \\ \xi_i^1 &\equiv \square \cdots \square 1 \square \cdots \square \end{aligned} \tag{8.8}$$

with the 0 or 1 in the ith position. It is clear that through intersections of these basic formae we can build all higher-order formae (again, exactly the higher-order schemata), and can identify individual solutions by noting the equivalence class to which they belong for each of the basic equivalence relations. Examples of this coding indicating the membership patterns of several formae are shown in table 8.2. Qualitatively we see that formae are neither local nor purely periodic in extent. Further, it is clear that as we increase the number of formae in order to approximate $\mathcal{S}$ more and more accurately, there is no clear notion of the limiting properties of the formae. This will be made evident in section 8.5 when we examine the limiting behaviour of the genetic operators derived from this representation.

Gray Coding

The principal motivation for considering Gray coding is the perceived problem of Hamming Cliffs with traditional integer coding. An example of a Hamming cliff is the transition from 7 to 8 in the traditional coding, where a relatively large number of bits change value with a small step in the search space (e.g. $\cdots 0111$ to $\cdots 1000$). The attraction of Gray coding (Caruana & Schaffer, 1988) is that the strings representing adjacent values always differ by exactly one bit. There are numerous possible mappings of the integers to binary strings that have this adjacency property, but the most commonly used one codes the integer x as the (binary) value of $x \oplus \lfloor x/2 \rfloor$ where $\oplus$ is the

Value	Binary	□□1	□0□	□01	Gray	□□1	□0□	□01	Dedekind	ξ_5^0	ξ_2^1	$\xi_5^0 \cap \xi_2^1$
0	000	-	•	-	000	-	•	-	0000000	•	-	-
1	001	•	•	•	001	•	•	•	0000001	•	-	-
2	010	-	-	-	011	•	-	-	0000011	•	•	•
3	011	•	-	-	010	-	-	-	0000111	•	•	•
4	100	-	•	-	110	-	-	-	0001111	•	•	•
5	101	•	•	•	111	•	-	-	0011111	-	•	-
6	110	-	-	-	101	•	•	•	0111111	-	•	-
7	111	•	-	-	100	-	•	-	1111111	-	•	-

Table 8.2: The figure shows the way in which elements of $\mathbb{Z}_n$ (used to approximate a real-valued interval) are represented using traditional binary coding, Gray coding, and the Dedekind representation developed here. Several example formae are illustrated in each case. Note that in the first two cases, formae correspond to schemata and are global in extent (including intersections), while in the third case, formae correspond to intervals on the line and encapsulate the notion of locality, presumably important in real optimisation. These representations are used to mathematically derive genetic operators suitable for computational implementation—it is *not* necessary (or even feasible, in the case of Dedekind) to store solutions in the forms shown here.

bitwise exclusive-or operator. If we write the binary value of x (the traditional integer coding) as $b_{k-1} \cdots b_1 b_0$ and the Gray-coded representation of i as $g_{k-1} \cdots g_1 g_0$ then we have the relationships

$$g_i = b_{i+1} \oplus b_i \tag{8.9}$$

and

$$b_i = b_{i+1} \oplus g_i, \tag{8.10}$$

which allow conversion from one form to the other (taking $b_k = 0$).

These relationships allow us to write the characterisation χ_{Gray} that generates the Gray-coding representation as the k equivalence relations

$$\psi_i(x, y) = \begin{cases} 1, & \text{if } (x \oplus \lfloor x/2 \rfloor) \otimes 2^i = (y \oplus \lfloor y/2 \rfloor) \otimes 2^i, \\ 0, & \text{otherwise.} \end{cases} \tag{8.11}$$

where $i \in \{0, 1, \ldots, k-1\}$. As with traditional integer coding, the formae can be exactly identified with (now Gray-coded) schemata. An example of the coding scheme along with the members of several formae are shown in table 8.2. Although the basic formae never contain 'singleton' solutions (with no immediate neighbours also in the formae), it is clear that they are still highly non-local in extent. It will also be shown that the limiting properties of the genetic operators (as the level of approximation is improved) are no better than the traditional coding.

8.3.3 Representations that "Capture" Continuity

The Dedekind Representation

We seek to characterise $\mathbb{Z}_n$ based on our beliefs about the structure of the problems we will be attacking. For the wide class of real-parameter optimisation problems, perhaps the simplest belief we might hold is some notion of continuity. That is, we believe that small changes in the parameter values will generally lead to small changes in the objective function. In evolution strategies, this belief is termed *the principle of strong causality*, and in real analysis functions with such a property are termed *Hölder continuous* or *Lipschitz*.

In order to quantify this belief, we must identify groups of solutions (formae) with related performance. To do this, we will characterise solutions based on locality using the idea of *Dedekind cuts*:

Definition (Dedekind cut). A Dedekind cut is a partitioning of the rational numbers into two non-empty sets, such that all the members of one are less than all those of the other. For example, the positive irrationals can be defined as Dedekind cuts on the positive rationals (e.g. $\sqrt{2} \triangleq \langle \{x \in \mathbb{Q}^+ \mid x^2 < 2\}, \{x \in \mathbb{Q}^+ \mid x^2 > 2\} \rangle$).
■

Our formal characterisation χ_{Dedekind} consists of n basic equivalence relations defining cuts on $\mathbb{Z}_n$:

$$\psi_i(x,y) = \begin{cases} 1, & \text{if } x,y \geq i \text{ or } x,y < i, \\ 0, & \text{otherwise,} \end{cases} \tag{8.12}$$

where $i \in \{1,\ldots,n-1\}$. These equivalence relations induce half-space equivalence classes of the following type:

$$\begin{aligned} \xi_i^0 &\equiv \{0,1,\ldots,i-1\}, \\ \xi_i^1 &\equiv \{i,i+1,\ldots,n-1\}. \end{aligned} \tag{8.13}$$

It is easy to see that intersections of these basic formae result in sets defining closed intervals, as illustrated in table 8.2. Note that, formally, this representation has $n-1$ highly non-orthogonal (constrained) binary genes coding a single approximated parameter, instead of the $k = \lceil \log_2 n \rceil$ orthogonal genes of traditional integer coding or Gray coding. However, we will see that the operators we derive from this representation and their limiting behaviour as we increase the level of approximation ($n \to \infty$) are much more natural with our new definitions.

While it is somewhat ironic that this formalism of the real representation utilises binary genes, we emphasise once again that we do *not* propose to store or manipulate solutions in this form, but only to apply our design principles to (mathematically rather than computationally) develop and analyse our genetic operators.

8.3.4 Extending the Representations to Multiple Parameters

It is a simple matter to extend any of the representations to higher dimensions by forming products of the one-dimensional equivalence relations. For example, in a two-dimensional search space, approximated by $\mathbb{Z}_n \times \mathbb{Z}_n$, the basic equivalence relations for the Dedekind representation are of the form

$$\psi_{ij}(x,y) \equiv \psi_i(x_1,y_1) \times \psi_j(x_2,y_2) = \begin{cases} 1, & \text{if } (x_1,y_1 \geq i \text{ or } x_1,y_1 < i) \\ & \text{and } (x_2,y_2 \geq j \text{ or } x_2,y_2 < j), \\ 0, & \text{otherwise,} \end{cases} \tag{8.14}$$

with equivalence classes of the form

$$\begin{aligned} \xi_{ij}^{00} \equiv \xi_i^0 \times \xi_j^0 &= \{(0,0),\ldots,(0,j-1),\ldots,(i-1,0),\ldots,(i-1,j-1)\}, \\ \xi_{ij}^{10} \equiv \xi_i^1 \times \xi_j^0 &= \{(i,0),\ldots,(i,j-1),\ldots,(n-1,0),\ldots,(n-1,j-1)\}, \\ \xi_{ij}^{01} \equiv \xi_i^0 \times \xi_j^1 &= \{(0,j),\ldots,(0,n-1),\ldots,(i-1,j),\ldots,(i-1,n-1)\}, \\ \xi_{ij}^{11} \equiv \xi_i^1 \times \xi_j^1 &= \{(i,j),\ldots,(i,n-1),\ldots,(n-1,j),\ldots,(n-1,n-1)\}. \end{aligned} \tag{8.15}$$

The traditional binary and Gray codings can be similarly extended.

The Isodedekind Representation

An alternative approach to extending the Dedekind representation to multiple parameters is to define a more general characterisation based on new equivalence relations. Because it seems somewhat artificial to characterise locality only along the axis directions, we might think of devising a characterisation to capture locality more generally. Thus, we could define $\chi_{\text{Isodedekind}}$ containing basic equivalence relations that partition $\mathbb{Z}_n^m$ using cut planes which have arbitrary orientation. This would require our conceptual chromosome to have alleles indicating on which side of a series of cuts it fell in *every* possible direction from the origin. Thus in the limiting case, the chromosome consists of a continuous infinity of continuous infinities of genes! We do not exhibit a precise discrete formulation here for reasons of space, but it is straightforward to visualise the limiting forms that the generalised operators take in such a case (see for example figure 8.5). The Isodedekind representation demonstrates convincingly that even extremely non-orthogonal characterisations can be used successfully.

8.4 Forma Variance Calculations

As discussed in chapter 6, one measure which indicates how well formae succeed in grouping together solutions of related fitness is mean forma variance. By generating random formae of a particular size and measuring the fitness variance within them, we can estimate the mean variance for formae of a given size. This was shown to be a good qualitative indicator of relative algorithmic performance (Radcliffe & Surry, 1994b).

For Dedekind representation, all formae are convex simplices $\mathbb{R}^m$ in bounded by hyper-planes perpendicular to the coordinate axes, while for the Isodedekind representation formae are convex simplices bounded by arbitrary hyper-planes. In order to investigate analytically the forma variance characteristics of these two new representations, we will consider specialised symmetric formae—hypercubes for the Dedekind representation and hyper-spheres for the Isodedekind representation.

Consider an objective function $f(\boldsymbol{x})$ of m real parameters. If f is continuous with bounded derivatives, then near a point $\boldsymbol{x}_0$ we can express f using Taylor's theorem:

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \boldsymbol{\nabla} f \cdot (\boldsymbol{x} - \boldsymbol{x}_0) + O(||\boldsymbol{x} - \boldsymbol{x}_0||^2) \tag{8.16}$$

so that as $||\boldsymbol{x} - \boldsymbol{x}_0|| \to 0$, f becomes essentially linear. We are interested in the typical variance observed in function values for formae of a specific size. For a forma ξ the variance is given by the expression

$$\mathrm{var}_\xi f = \frac{\int_\xi (f(\boldsymbol{x}) - \overline{f}_\xi)^2 d\boldsymbol{x}}{\int_\xi d\boldsymbol{x}} \tag{8.17}$$

which we proceed to calculate for the two representations.

Dedekind

For the Dedekind representation we take ξ to be the hypercube $|x_i| \leq \delta$, with volume $\mathrm{Vol}(\xi) = (2\delta)^m$. Without loss of generality, take $\boldsymbol{x}_0 = \mathbf{0}, f(\mathbf{0}) = 0$ (so that $\overline{f}_\xi = 0$), and $\boldsymbol{\nabla} f(\mathbf{0}) = c \sum_i \alpha_i \boldsymbol{e}_i$ where $\sum \alpha_i^2 = 1$ (i.e. $c = ||\boldsymbol{\nabla} f(\mathbf{0})||$). Then if $c \neq 0$, as $\delta \to 0$ we have

$$\begin{aligned}
\mathrm{var}_\xi f &= \int_{-\delta}^{\delta} \cdots \int_{-\delta}^{\delta} \left(c \sum_{i=1}^{m} \alpha_i x_i \right)^2 dx_1 \cdots dx_m \Bigg/ \int_{-\delta}^{\delta} \cdots \int_{-\delta}^{\delta} dx_1 \cdots dx_m \\
&= \frac{c^2}{(2\delta)^m} \int_{-\delta}^{\delta} \cdots \int_{-\delta}^{\delta} \left(\sum_{i=1}^{m} \alpha_i^2 x_i^2 + 2 \sum_{i<j} \alpha_i \alpha_j x_i x_j \right) dx_1 \cdots dx_m \\
&= \frac{c^2}{(2\delta)^m} \left(\sum_{i=1}^{m} \alpha_i^2 \int_{-\delta}^{\delta} x_i^2 dx_1 \cdots dx_m + 2 \sum_{i<j} 0 \right) \\
&= \frac{c^2}{(2\delta)^m} \left(\sum_{i=1}^{m} \alpha_i^2 \; x_i^3/3 \Big|_{-\delta}^{\delta} (2\delta)^{m-1} \right) \\
&= \frac{c^2}{12} \sqrt[m]{\mathrm{Vol}(\xi)}^2 .
\end{aligned} \tag{8.18}$$

Isodedekind

For the Isodedekind representation, we take ξ to be the hyper-sphere $||\boldsymbol{x}|| < \delta$. We will make use of the result

$$V_m(r) = \frac{\pi^{\lfloor \frac{m}{2} \rfloor} 2^{\lfloor \frac{m+1}{2} \rfloor}}{m(m-2)(m-4)\cdots} r^m \tag{8.19}$$

where $V_m(r) = \mathrm{Vol}(\xi)$ is the volume of an m-dimensional sphere of radius r. (This result can be established inductively using the integral relations $I_{2k} = \int_{-r}^{r}(r^2-x^2)^{2k}dx = 2^{2k+1}(k!)^2 r^{2k+1}/(2k+1)!$ and $I_{2k+1} = \int_{-r}^{r}(r^2-x^2)^{(2k+1)/2}dx = (2k+1)!\pi r^{2k+2}/(2^{2k+1}k!(k+1)!)$.) We will also use the defining relationship $V_m(r) = \int_{-r}^{r} V_{m-1}(\sqrt{r^2-x^2})dx = V_{m-1}(1)\int_{-r}^{r}(r^2-x^2)^{(n-1)/2}dx$.

As before then, without loss of generality we take $\boldsymbol{x}_0 = \mathbf{0}$, $f(\mathbf{0}) = 0$, and now $\boldsymbol{\nabla} f(\mathbf{0}) = c\boldsymbol{e}_1$ (as the hyper-sphere is clearly invariant to rotation of the coordinate axes). Then for $c \neq 0$,

$$\begin{aligned}
\mathrm{var}_\xi f &= \int_\xi (cx_1)^2 \boldsymbol{dx} \Big/ \int_\xi \boldsymbol{dx} \\
&= \int_{-\delta}^{\delta} (cx_1^2) V_{m-1}\left(\sqrt{\delta^2 - x_1^2}\right) dx_1 / V_m(\delta) \\
&= c^2 \frac{V_{m-1}(1)}{V_m(\delta)} \int_{-\delta}^{\delta} x^2 (\delta^2 - x^2)^{\frac{m-1}{2}} dx \\
&= c^2 \frac{V_{m-1}(1)}{V_m(\delta)} \int_{-\delta}^{\delta} \frac{(\delta^2 - x^2)^{\frac{m+1}{2}}}{m+1} dx \quad \text{(integrating by parts)} \\
&= \frac{c^2 V_{m-1}(1) V_{m+2}(\delta)}{(m+1) V_m(\delta) V_{m+1}(1)} \qquad (8.20)
\end{aligned}$$

Using $V_{m+2}(1)/V_m(1) = 2\pi/(m+2)$, and $\delta = \sqrt[m]{V_m(\delta)/V_m(1)}$ we see that:

$$\begin{aligned}
\mathrm{var}_\xi f &= \frac{c^2 2\pi(m+1)}{(m+1)(m+2)2\pi} \delta^2 \\
&= \frac{c^2}{(m+2)\sqrt[m]{V_m(1)^2}} \sqrt[m]{\mathrm{Vol}(\xi)}^2 . \qquad (8.21)
\end{aligned}$$

Comparing with (8.18), we see that the forma variance for the Isodedekind representation is strictly less than that of the Dedekind representation except in the case $m = 1$ where equality holds.

We can use Stirling's formula, $n! \simeq \sqrt{2\pi n}(n/e)^n$, to consider the limiting case

$$\begin{aligned}
\lim_{m\to\infty} \sqrt[m]{V_m(1)^2} &= \lim_{m\to\infty} \left(\frac{\pi^{\lfloor \frac{m}{2} \rfloor} 2^{\lfloor \frac{m+1}{2} \rfloor}}{m(m-2)(m-4)\cdots} 1^m \right)^{\frac{2}{m}} \\
&= \lim_{m\to\infty} \left(\pi^{\lfloor \frac{m}{2} \rfloor} / \left(\frac{m}{2}\right)! \right)^{\frac{2}{m}} \quad (m \text{ even for convenience}) \\
&= \lim_{m\to\infty} \left(\frac{1}{\pi m}\right)^{\frac{1}{m}} \left(\frac{2\pi e}{m}\right)^{\frac{m}{2}\frac{2}{m}} \qquad (8.22)
\end{aligned}$$

so that for formae of fixed volume:

$$\begin{aligned}
\lim_{m\to\infty} \mathrm{var}_\xi f &= \lim_{m\to\infty} \frac{c^2 m}{2\pi e(m+2)} (\pi m)^{\frac{1}{m}} \sqrt[m]{\mathrm{Vol}(\xi)}^2 \\
&= \frac{c^2}{2\pi e} \sqrt[m]{\mathrm{Vol}(\xi)}^2 . \qquad (8.23)
\end{aligned}$$

To summarise numerically, then, we have

m	$\text{var}_\xi f$
1	$0.0833c^2 \sqrt[1]{\text{Vol}(\xi)}^2$
2	$0.0796c^2 \sqrt[2]{\text{Vol}(\xi)}^2$
3	$0.0770c^2 \sqrt[3]{\text{Vol}(\xi)}^2$
...	...
∞	$0.0586c^2 \sqrt[\infty]{\text{Vol}(\xi)}^2$

Expected Forma Variance

Note that for both the Dedekind and Isodedekind representation we have

$$\text{var}_\xi f = c(\xi)^2 k_m \sqrt[m]{\text{Vol}(\xi)}^2 \tag{8.24}$$

with $c(\xi) = ||\nabla f(\xi)||^2$, the (constant) gradient in the forma ξ, and k_m a constant depending only on the dimension of f and the shape of formae under consideration. Over a set of formae Ξ, then, the expected value of the forma variance can be simply expressed as:

$$\text{Exp}_\Xi\{\text{var}_\xi f\} = \text{Exp}_\Xi\{c(\xi)^2\} k_m \sqrt[m]{\text{Vol}(\xi)}^2 \tag{8.25}$$

For instance, in the case of formae of fixed volume drawn uniformly at random from a region R, $\text{Exp}_\Xi\{c(\xi)^2\} = \text{Exp}_R\{||\nabla f||^2\}$, which we validate empirically in section 8.6.

Relationship to Lipschitz Criterion

Building from the results above, it is straightforward to show that requiring the forma variance to decrease to zero as forma volume decreases is closely related to making a Lipschitz assumption on the objective function. For illustrative purposes, we consider the one-dimensional case in which formae become intervals (although it is not difficult to extend the results to higher dimensions). As above, the fitness variance of a function $f(x)$ over a forma ξ of width δ, centred at $x = x'$ can be written as:

$$\text{var}_\xi f = \frac{1}{\delta} \int_{x'-\delta/2}^{x'+\delta/2} (f(x) - \bar{f}_\xi)^2 dx. \tag{8.26}$$

Now, assume that f is Lipschitz over an interval $[\alpha, \beta]$, that is,

$$x, y \in [\alpha, \beta] \Longrightarrow |f(x) - f(y)| < \varepsilon |x - y| \tag{8.27}$$

for some $\varepsilon \in \mathbb{R}^+$. For any forma ξ of size δ contained in $[\alpha, \beta]$, we know by the the mean value theorem that $\bar{f}_\xi = f(c)$ for some $c \in [t - \delta/2, t + \delta/2]$. We then have

$$\begin{aligned} \text{var}_\xi f &< \frac{1}{\delta} \int_{t-\delta/2}^{t+\delta/2} (\varepsilon(x - c))^2 dx \\ &= \frac{\varepsilon^2}{3\delta} (x - c)^3 \Big|_{t-\delta/2}^{t+\delta/2} \end{aligned}$$

$$\leq \frac{\varepsilon^2\delta^2}{3}$$
$$= \frac{\varepsilon^2}{3}\sqrt[m]{\mathrm{Vol}(\xi)}^2 \tag{8.28}$$

Conversely, it is trivial to show that if f is not Lipschitz on $[\alpha, \beta]$, then there must be a limiting sequence of formae within the interval for which

$$\lim_{\mathrm{Vol}(\xi)\to 0} \mathrm{var}_\xi f = \infty \tag{8.29}$$

Thus the notion that formae should group together solutions of related fitness has been shown to be closely linked to the characterisations on which we based the construction of the Dedekind and Isodedekind representations—namely that the functions of interest were in some sense smooth (satisfying a Lipschitz condition, or more loosely the principle of strong causality).

For the traditional integer coding and Gray codings used with genetic algorithms, there is no analogous limiting behaviour that can be extracted. We might speculate on some relationship to periodic behaviour of the function (perhaps with respect to its discrete Fourier coefficients?), but it is not simply that. It has been "discovered" several times that simply by shifting the origin or rescaling the axes, the behaviour of algorithms based on these representation can change radically. This is clearly very undesirable behaviour.

Fundamentally, we argue that this stems from the lack of a principled foundation for the traditional representations—they do not encapsulate particular beliefs about the problem domains of interest. In fact, many workers have taken the diametrically opposite approach of trying to discover what it is that these representations *are* characterising, for instance by constructing specialised functions over the integers which lead the algorithms in particular directions. For instance, consider the behaviour of a trap or long-path function (Horn *et al.*, 1994) as we take the limit $n \to \infty$. The trap function becomes analogous to a superposition of a simple linear function with a Dirac delta function whilst it is not clear whether the long-path function can be said to have any sensible limit. While the study of such functions discretised for fixed values of n may be intrinsically interesting, it is far from clear that it bears on the problem of how algorithms based on these representations behave for practical real-parameter functions.

8.5 Derivation of Genetic Operators

It is straightforward to derive forms of the generalised genetic operators described in chapter 5, as summarised in table 8.1.

For the traditional binary and Gray codings, the generalised operators reduce to "standard" forms, since the representations are orthogonal (all combination of allele

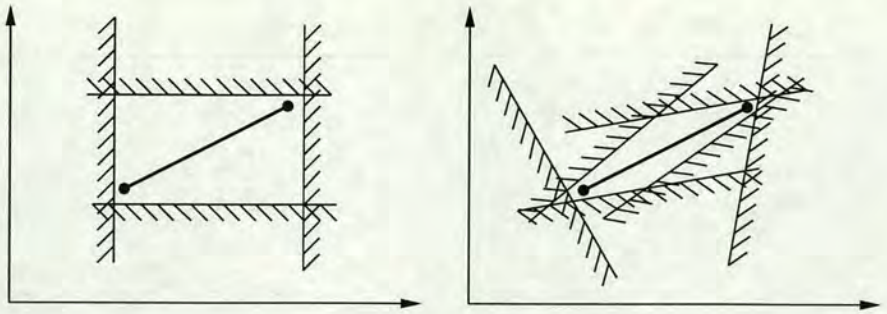


Figure 8.5: The figure illustrates two problem-specific forms of the random respectful recombination (R^3) operator for a two-dimensional real-parameter optimisation problem. (a) On the left, the Dedekind representation is defined by cut formae along each axis. Since R^3 requires that the child be a member of all formae to which both parents belong, the child must lie within the hypercube (square here) with its two parents at opposite vertices. Hence R^3 reduces to BLX-0 in this case. (b) On the right, the Isodedekind representation is defined by cut formae in arbitrary directions. Thus R^3 requires, in the limit, that the child lies on the line segment bounded by the two parents, so that R^3 is equivalent to line recombination in this case. It is also instructive to compare algorithms that incorporate R^3 based on these different representations—see figure 8.1.

values are legal). Thus, RAR, RTR and R^3 reduce to uniform crossover, GNX reduces to N-point crossover, and BMM reduces to bitwise point mutation.

For the Dedekind representation, it is clear that both R^3 and RTR require that the child be uniformly selected to lie in the box determined by the parents (see also figure 8.5). Thus, in one dimension, the crossover operator $\mathcal{X} : \mathbb{Z}_n \times \mathbb{Z}_n \times [0,1) \longrightarrow \mathbb{Z}_n$ acting on parents x and y (without loss of generality, $x < y$) and control parameter $\kappa \in [0,1)$ results in the child $\mathcal{X}(x,y,\kappa) = x + \lfloor \kappa(y-x+1) \rfloor$. It is clear that in the limit of $n \to \infty$, this is equivalent to BLX-0 as defined by Eshelman & Schaffer (1992); see figure 8.2. For this representation, it is not difficult to see that RAR is also equivalent to BLX-0 (it is easy to show that the child must lie in the interval defined by the parents, and only slightly more difficult to demonstrate that the likelihood is uniform over the interval).

For the Isodedekind representation, R^3 and RTR reduce to line recombination as shown in figure 8.5, and RAR simply requires that we are able to generate any point in the search space.

Turning to mutation, if we analyse our representation-independent mutation operator BMM with the Dedekind representation, it is clear that a minimal mutation involves flipping the value of one of the two bits forming the transition from ones to zeros in the genome. Thus a fixed-length sequence of minimal mutations is equivalent to a random walk away from the original transition point. We will show that this reduces in the limit of $n \to \infty$ to standard gaussian creep mutation with width parameterised by the gene-wise mutation probability.

Proof: Assume that each parameter is represented by n genes in the Dedekind

representation. Consider the action of BMM on a single parameter (as it is clear that genes in different parameters are orthogonal). Given a gene-wise mutation rate p_M, and an effective chromosome length ℓ_n (which we will find must be proportional to n^2), we make a binomial number s of minimal mutations, where s is chosen from the distribution $S \sim B(\ell_n, p_M)$. Now, we know that the binomial distribution $B(n,p)$ is asymptotically approximated by the normal distribution $N(np, \sqrt{np(1-p)})$ provided that $\sqrt{np(1-p)} \ll np$. In our case, we take p_m fixed, and $\ell_n \to \infty$ as $n \to \infty$ so that S is asymptotically approximated by $S \sim N(\ell_n p_M, \sqrt{\ell_n p_M(1-p_M)})$. Now, each minimal mutation results in a step of size $\Delta = L/n$ to the left or right of the current value, where L is the length of the interval in which the parameter lies. Thus once we have chosen a number of steps s, distributed according to $S \sim N(\ell_n p_M, \sqrt{\ell_n p_M(1-p_M)})$ we perform a random walk with step length Δ, so that the final displacement x is distributed conditionally on s, according to the normal distribution $X|S \sim N(0, \sqrt{s}L/n)$. Further, we can thus write the unconditional p.d.f. for X as:

$$p(x) = \int_{-\infty}^{\infty} p(x|s)p(s)ds. \tag{8.30}$$

Consider now the moment-generating function for X, $m_X(t)$, given by:

$$\begin{aligned} m_X(t) = E(e^{tx}) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x|s)p(s)e^{tx}dsdx \\ &= \int_{-\infty}^{\infty} p(s)\int_{-\infty}^{\infty} p(x|s)e^{tx}dxds \\ &= \int_{-\infty}^{\infty} p(s)m_{X|S}(t)ds. \end{aligned} \tag{8.31}$$

Now, since $X|S \sim N(0, \sqrt{s}L/n)$, a normal distribution, we know that $m_{X|S}(t) = \exp(0t + (\sqrt{s}L/n)^2t^2/2)$, which yields

$$\begin{aligned} m_X(t) &= \int_{-\infty}^{\infty} p(s)\exp\left(\frac{s}{2}\left(\frac{Lt}{n}\right)^2\right)ds \\ &= m_S\left(\frac{1}{2}\left(\frac{Lt}{n}\right)^2\right). \end{aligned} \tag{8.32}$$

Finally, since $S \sim N(\ell_n p_M, \sqrt{\ell_n p_M(1-p_M)})$ we have $m_S(t) = \exp(\ell_n p_M t + \ell_n p_M(1-p_M)t^2/2)$, so that

$$m_X(t) = \exp\left(\frac{p_M L^2 \ell_n}{2n^2}t^2 + \frac{p_M(1-p_M)L^4\ell_n}{8n^4}t^4\right). \tag{8.33}$$

We thus choose our length scale $\ell_n = n^2$ to make the limit finite, and see that $\lim_{n\to\infty} m_X(t) = \exp(p_M L^2 t^2/2)$. This is simply the moment generating function for $N(0, \sqrt{p_M}L)$ so the final displacement X must have the identical distribution. Thus BMM instantiated for the Dedekind representation is that operator which adds gaussian noise with width $\sqrt{p}_M L$ to each parameter.

Note that for the Isodedekind representation a sequence of minimal mutations involves a random walk with steps taken in arbitrary directions in $\mathbb{R}^m$. Thus BMM is likely to be equivalent to Gaussian mutation, but the precise scaling factors have not yet been derived.

8.6 Search Strategies

These operators can then be used to instantiate a fixed representation-independent algorithm for each of the four representations presented here. The performance of the resulting search strategies is illustrated for Schaffer's two-dimensional F6 function in figure 8.1. In his work, $-100 \leq x_1, x_2 \leq 100$, and

$$f_6(x_1, x_2) \triangleq 0.5 - \frac{\sin^2 \sqrt{x_1^2 + x_2^2} - 0.5}{1.0 + 0.001(x_1^2 + x_2^2)^2} \tag{8.34}$$

with the maximal value of 1 occurring at $(x_1, x_2) = (0, 0)$.

Following the ideas in chapter 6, we study the forma variance properties of this function for the four representations in figure 8.6. Although the figure is specific to the two-dimensional function considered here, it is clear that the bulk of the behaviour shown in the figure is generic—for any real-parameter function with bounded derivatives there will be some length-scale below which the observed behaviour will occur for the Dedekind representations (exactly as predicted in section 8.4), and for the binary representations the same near-random behaviour is to be expected as the representation does not capture the essential structure of the function being optimised.

8.7 Discussion

This chapter has presented formal constructions for two genetic representations for real-parameter optimisation based on characterising the notion of locality—the Dedekind and Isodedekind representations. Generalised genetic operators are shown to reduce in these representations to precisely those operators which are widely used in practice, namely blend crossover, line recombination and gaussian creep mutation. Both of these representations are highly non-orthogonal, but this is seen not to present difficulties. Analysis of the limiting behaviour of discrete approximations to a continuous search space has shown that the more traditional binary and Gray codings have pathological behaviour. This is illustrated qualitatively when a fixed algorithm is instantiated for all four representations and radically different behaviour is observed (figure 8.1).

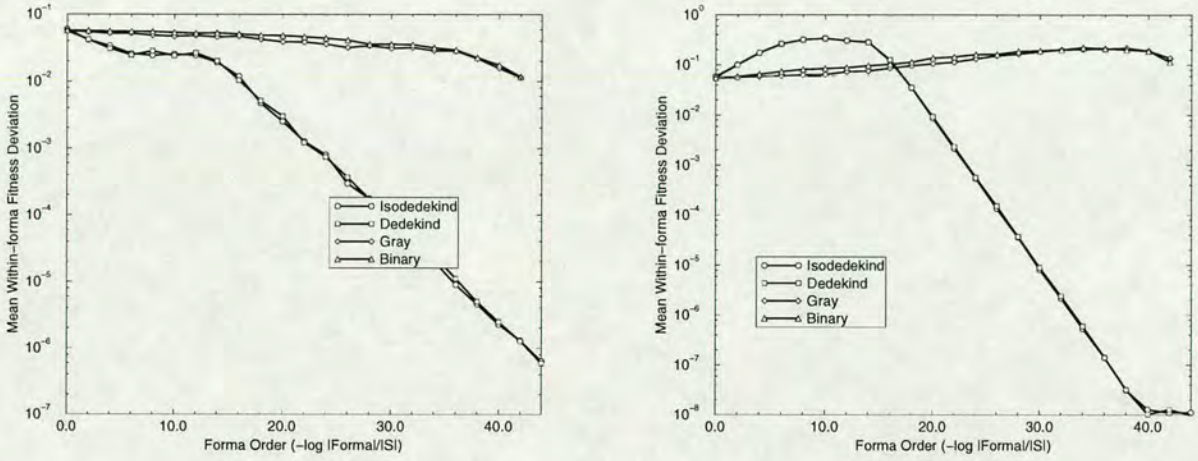


Figure 8.6: The graphs show the mean standard deviation of fitness within randomly generated formae (left), and randomly generated formae containing the optimal solution (right), plotted against mean log forma size for each of the four representations considered in this chapter. Each plotted point is based on 200 sampled tours drawn at random from each of 200 formae of fixed order, for Schaffer's F6 function. Confidence intervals (standard errors) are omitted as they are smaller than the plotted symbols. The Dedekind and Isodedekind representations show a significantly faster reduction in variance as formae size decreases—indeed for the binary representations variance actually increases as forma size decreases if formae containing the optimum are considered (intuitively this is because such formae essentially contain a few near-optimal solutions with fitness near 1 and a number of essentially random points whose fitness will be near 0.5). Although the curves for the Dedekind and Isodedekind representations are very similar, by fitting a least-squares regression to the linear region of the left-hand graph, we calculate fitted slopes of $0.151 \simeq 0.5 \log_{10} 2$, and a difference in intercepts of $0.016 \simeq 0.5(\log_{10} 12 - \log_{10} 4\pi)$ as predicted by the results of section 8.4. The curves for formae containing the optimum exhibit a steeper slope because the linear term in the Taylor expansion vanishes, leaving a quadratic dependence.

Chapter 9

Memetic Algorithms: Genetic Algorithms incorporating Local Search

In the end we all come to be cured of our sentiments. Those whom life does not cure death will. The world is quite ruthless in selecting between the dream and the reality, even where we will not. Between the wish and the thing the world lies waiting. (Cormac McCarthy)

The rôle of local search in the context of genetic algorithms and the wider field of evolutionary computing has been much discussed. The traditional view, which can be traced back to Holland (1975), has been that the primary search operator in evolutionary computing should be recombination. In its most extreme form, this view casts mutation and other local operators as mere adjuncts to recombination, playing auxiliary (if important) rôles such as keeping the gene pool well stocked and helping to tune final solutions. There have, however, long been advocates of a greater rôle for mutation, hill-climbing and local refinement. The arguments for serious consideration of operators other than recombination for primary search come in many forms and are inspired by widely differing applications. For example, Davis (1991b) advocates *hybridisation* of genetic algorithms with domain-specific techniques for "real world" optimisation, by incorporating extra move operators. He regularly uses sophisticated decoders that make use, for example, of greedy algorithms and repair mechanisms. Ackley (1987) recommends *genetic hill-climbing,* in which crossover plays a rather less dominant rôle. Mühlenbein (1992) argues theoretically and Gorges-Schleuter (1989) provides empirical demonstrations that local search can play a key rôle, and Mühlenbein (1989) incorporates it as a fundamental component of his particular notion of a parallel genetic algorithm with a structured population. Meanwhile, the *Evolution Strategies* community has always placed more emphasis on mutation than crossover (Bäck *et al.*, 1991). Countless other advocates of a greater emphasis on non-recombinative elements of evolutionary search could be cited, especially from the ranks of those competing with

domain-specific techniques.

Moscato & Norman (1992) have introduced the term *memetic algorithm* to describe evolutionary algorithms in which local search plays a significant part. This term is motivated by Richard Dawkins's notion of a *meme* as a unit of information that reproduces itself as people exchange ideas (Dawkins, 1976). A key difference exists between genes and memes: before a meme is passed on, it is typically adapted by the person who transmits it as that person thinks, understands and processes the meme, whereas genes get passed on unchanged. Moscato and Norman liken this adaptation to local refinement, and therefore promote the term "memetic algorithm" to describe genetic algorithms that use local search heavily.

The purpose of this chapter is to formalise Norman and Moscato's memetic algorithms within the framework presented in chapter 5. We use forma analysis (Radcliffe, 1991a; 1994) to define a general hill-climbing operator to support the construction of this representation-independent (formal) memetic algorithm. This formalism is then demonstrated in practice by application to the travelling sales-rep problem (TSP) discussed earlier in chapter 7.

9.1 Memetic Algorithms

Our first task is to provide a consistent formal framework for considering memetic and genetic algorithms. Informally, the idea exploited to achieve this is that if a (true) local optimiser is added to a genetic algorithm, and applied to every child before it is inserted into the population (including the initial population) then a memetic algorithm can be thought of simply as a special kind of "genetic" search over the subspace of local optima (figure 9.1). Recombination and mutation will usually produce solutions that are outside this space of local optima (and can thus be regarded as "damaged") but a local optimiser can then "repair" such solutions to produce final children that lie within this subspace, yielding a memetic algorithm. Section 9.1.1 formalises these notions and section 9.1.2 discusses when such memetic search might be more appropriate than genetic search.

9.1.1 Formal Memetic Algorithms

Using the notation of chapter 3, consider a search space $\mathcal{S}$ (of *phenotypes*) and a representation space $\mathcal{C}$ (of *genotypes*), with growth function g. Let f be the fitness function, which it will be convenient to regard as a mapping

$$f : \mathcal{C} \longrightarrow \mathbb{R}^+. \tag{9.1}$$

It will be assumed without loss of generality that the aim is to maximise fitness, and the set of global optima will be denoted $\mathcal{C}^\star \subset \mathcal{C}$.

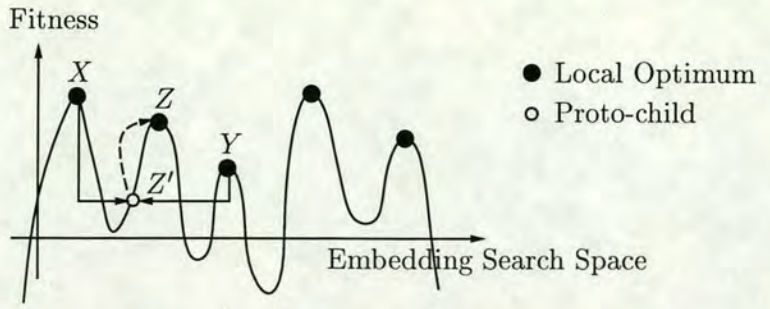


Figure 9.1: Memetic algorithms search over the subspace of local optima within the embedding search space of all solutions. After recombination, the proto-child typically lies outside this subspace and a local optimiser is used to "repair" the child so that it lies at a local optimum. Here parents X and Y produce the proto-child Z', which is then optimised to produce the final child Z.

Let ω be a stochastic unary move operator over $\mathcal{C}$. It will be convenient for the moment to accommodate the stochastic element of such an operator through a *control set,* $\mathcal{K}_\omega$, from which a *control parameter* will be drawn to determine which of the (typically many) possible moves actually occurs. For example, in the case of mutation of binary strings, a binary mask might be used as the control parameter with the presence of a 1 at position i indicating that the ith bit should be mutated. The functional form for ω will then be

$$\omega : \mathcal{C} \times \mathcal{K}_\omega \longrightarrow \mathcal{C}. \tag{9.2}$$

A chromosome $x \in \mathcal{C}$ will be said to be *locally optimal with respect to* ω, or ω*-opt,* if no chromosome of higher fitness than x can be generated from it by a single application of ω, i.e. if and only if

$$\forall \kappa \in \mathcal{K}_\omega : f \circ g(\omega(x, \kappa)) \leq f \circ g(x). \tag{9.3}$$

Let $\mathcal{C}_\omega \subset \mathcal{C}$ be the set of ω-opt chromosomes in $\mathcal{C}$, i.e.

$$\mathcal{C}_\omega \triangleq \{x \in \mathcal{C} \mid x \text{ is } \omega\text{-opt}\}. \tag{9.4}$$

A genetic algorithm applied to the task of optimising f over $\mathcal{C}$ has some goal such as finding some or all optima in $\mathcal{C}^\star$ or making rapid improvements towards fitter chromosomes. It is clear that for any move operator ω, all chromosomes in $\mathcal{C}^\star$ are ω-opt, and thus $\mathcal{C}^\star \subset \mathcal{C}_\omega$. In principle, it would thus be perfectly satisfactory to formulate the search instead over $\mathcal{C}_\omega$.

Given a representation space $\mathcal{C}$, a move operator ω, and the subspace $\mathcal{C}_\omega$ of local optima as above, define a *hill-climber* to be any stochastic, parameterised operator that, given a chromosome $x \in \mathcal{C}$, returns a local optimum in $\mathcal{C}_\omega$. Thus a hill-climber $\mathcal{H}$ (with respect to ω) with control set $\mathcal{K}_\mathcal{H}$ is any function

$$\mathcal{H} : \mathcal{C} \times \mathcal{K}_\mathcal{H} \longrightarrow \mathcal{C}_\omega. \tag{9.5}$$

Notice that there is no requirement that the solution returned be in any sense "near" the starting solution, though of course this will often be the case in practice.

Typical genetic algorithms produce new chromosomes by recombination of two parents followed by some small level of mutation, so that if

$$\mathcal{X} : \mathcal{C} \times \mathcal{C} \times \mathcal{K}_{\mathcal{X}} \longrightarrow \mathcal{C} \tag{9.6}$$

is the recombination operator (with control set $\mathcal{K}_{\mathcal{X}}$), and

$$\mathcal{M} : \mathcal{C} \times \mathcal{K}_{\mathcal{M}} \longrightarrow \mathcal{C} \tag{9.7}$$

is the mutation operator (with control set $\mathcal{K}_{\mathcal{M}}$), the combined genetic reproductive function $\mathfrak{R}_g$ would typically be given by the composition of mutation and recombination, $\mathfrak{R}_g = \mathcal{M} \circ \mathcal{X}$, yielding

$$\mathfrak{R}_g : \mathcal{C} \times \mathcal{C} \times \mathcal{K}_{\mathcal{X}} \times \mathcal{K}_{\mathcal{M}} \longrightarrow \mathcal{C}, \tag{9.8}$$

defined by

$$\mathfrak{R}_g(x, y, \kappa_{\mathcal{X}}, \kappa_{\mathcal{M}}) \triangleq \mathcal{M}(\mathcal{X}(x, y, \kappa_{\mathcal{X}}), \kappa_{\mathcal{M}}). \tag{9.9}$$

If, however, $\mathfrak{R}_g$ is further composed with a hill-climber $\mathcal{H}$ (with respect to some unary move operator ω), and restricted to $\mathcal{C}_\omega$, a memetic reproduction function $\mathfrak{R}_m \triangleq \mathcal{H} \circ \mathcal{M} \circ \mathcal{X}$ results:

$$\mathfrak{R}_m : \mathcal{C}_\omega \times \mathcal{C}_\omega \times \mathcal{K}_{\mathcal{X}} \times \mathcal{K}_{\mathcal{M}} \times \mathcal{K}_{\mathcal{H}} \longrightarrow \mathcal{C}_\omega, \tag{9.10}$$

defined by

$$\mathfrak{R}_m(x, y, \kappa_{\mathcal{X}}, \kappa_{\mathcal{M}}, \kappa_{\mathcal{H}}) \triangleq \mathcal{H}(\mathcal{M}(\mathcal{X}(x, y, \kappa_{\mathcal{X}}), \kappa_{\mathcal{M}}), \kappa_{\mathcal{H}}). \tag{9.11}$$

9.1.2 Decomposable Fitness Functions

While the general question of when it might be appropriate to use a memetic algorithm in preference to a genetic algorithm is beyond the scope of this chapter, one special situation can be considered that seems likely to be relatively favourable to the memetic variety. This arises when the fitness function is *decomposable,* in the sense that computing the fitness of a solution given the fitness of another solution that is "close" to it in some well-defined way (in the sense, informally, of having much genetic material in common with it) is significantly less computationally expensive than computing the fitness of a solution "from scratch". In the TSP, for example, computing the length of a tour resulting from a single subtour inversion to a tour whose length is already known is very much cheaper than computing the length of a general tour, so the fitness function is in that case decomposable. Contrariwise, when solving a system of non-linear equations, for example to compute the flow of gas through a pipe network, a small change in the chromosome can often have global effects and therefore computing the fitness of a chromosome is made no easier by knowing that of another similar chromosome. Given

that in most real-world optimisation problems calculation of fitness accounts for almost all the time spent in a genetic algorithm, it seems likely that memetic algorithms will be at a relative advantage when the fitness function is decomposable, provided that the moves it makes while hill-climbing are "small".

9.2 Representation-Independent Hill Climbing

As discussed in chapter 5, the principal complication that arises in defining representation-independent operators is that some combinations of gene values may be incompatible. While this is most obviously a problem for recombination operators, it is also a serious consideration for other move operators. Here we turn our attention to generalised memetic operators based on the representation-independent *patching* discussed in chapter 5 and representation-independent *hill-climbing* operators described below.

In section 9.1.1 a hill-climber (with respect to a move operator ω) was defined to be any operator $\mathcal{H}$ having the functional form given in equation 9.5:

$$\mathcal{H} : \mathcal{C} \times \mathcal{K}_{\mathcal{H}} \longrightarrow \mathcal{C}_{\omega}. \tag{9.5 bis}$$

It is easy to construct a hill-climber from ω by repeatedly applying ω to the chromosome to be optimised, cycling through all the control parameters from $\mathcal{K}_{\omega}$. There is considerable freedom in exactly how such a hill-climber operates. In particular, there are many ways to decide when to accept a move and in which order to cycle through the control parameters. The hill-climber constructed here will be *greedy,* that is, it will accept any improvement generated by the operator immediately (and will never backtrack), as opposed, for example, to testing all control parameters and then accepting the move that generates the biggest improvement. Note that in general there is no requirement that $\mathcal{H}$ accept only improving moves, so long as it terminates in a local optima (e.g. a hill-climber based on simulated annealing).

The order in which to test the control parameters in $\mathcal{K}_{\omega}$ is more open. Any order will suffice provided that all parameters are tested (preferably without repetition) but a fixed order will afford the operator considerably less freedom than a random order. Testing the parameters in a totally random order (excluding only repetition) is by far the most appealing theoretically, and will almost certainly show the best performance because it minimises the correlations between applications of $\mathcal{H}$. In practice, however, this requires the generation of a very large number of (pseudo-) random numbers, and maintenance of a list of the moves that have been tested (or of those that remain to be tried). Nevertheless, this form of hill-climbing is sufficiently important to be named, and will be referred to as *ideal greedy hill-climbing.*

Many compromises between a totally random and a fixed order of sampling $\mathcal{K}_{\omega}$ across applications of ω are possible. Two in particular will be considered. The first

is to construct a random permutation of the control parameters to ω when the hill-climber is invoked, and to sample these in sequence. When an application of ω yields an improvement (and is therefore accepted), a new starting point within the permuted values is chosen, but the parameters are then sampled in the same order. This scheme will be called *rotated cyclic greedy hill-climbing.* A minor augmentation of this scheme involves also exchanging a randomly chosen pair of control parameters in the permutation when a move is accepted. This scheme will be called *rotated transposed cyclic greedy hill-climbing.* Clearly there are endless variations on this theme, particularly when drawing on ideas from simulated annealing and tabu search.

For the purposes of the formal memetic algorithm that is the subject of this chapter, the move operator defining local optimality will be minimal mutation ($\hat{\mathcal{M}}$).

9.3 Case Study: Application to the TSP

Earlier sections having constructed the formal components required for a memetic algorithm, the present section seeks to construct an instantiation for comparison with its genetic counterpart and for simple experimentation with its coarser parameters. The problem tackled will be the travelling sales-rep problem (TSP) because this has a decomposable fitness function (and should therefore be favourable to memetic search), is well-known, and is relatively hard for evolutionary techniques. Most studies of large TSP instances have previously concluded that augmentation with local search is essential (e.g. Verhoeven *et al.*, 1992; Gorges-Schleuter, 1989), and the aim here is not to achieve good performance as such but rather to understand how well an unembellished implementation of a formal memetic algorithm can work in this problem domain. For this reason, a modest but non-trivial TSP instance is used for these experiments.

Previous studies by Whitley *et al.* (1989) and Radcliffe & Surry (1994b) have provided evidence, both theoretical and empirical, that undirected edges are a relatively suitable basis for a representation of this problem, so the undirected edge representation discussed in chapter 7 will be used.

9.3.1 Experimental Results

Empirical studies were undertaken using the Reproductive Plan Language RPL2 (Surry & Radcliffe, 1994b) running on super-scalar SPARC processors. The problem instance used for the first experiments was the 100-city Krolak 'C' problem from TSPLIB (Reinelt, 1990). These experiments used a panmictic population of size 100 with elitism, non-generational ("one-at-a-time") update, binary tournament selection with parameter 0.7, binary tournament replacement with parameter 0.7, recombination with probability 1.0 and mutation probability p_M of 0.02. The weight used for RAR was 2.0, and the number of cross points used for GNX was 2. Experiments were conducted us-

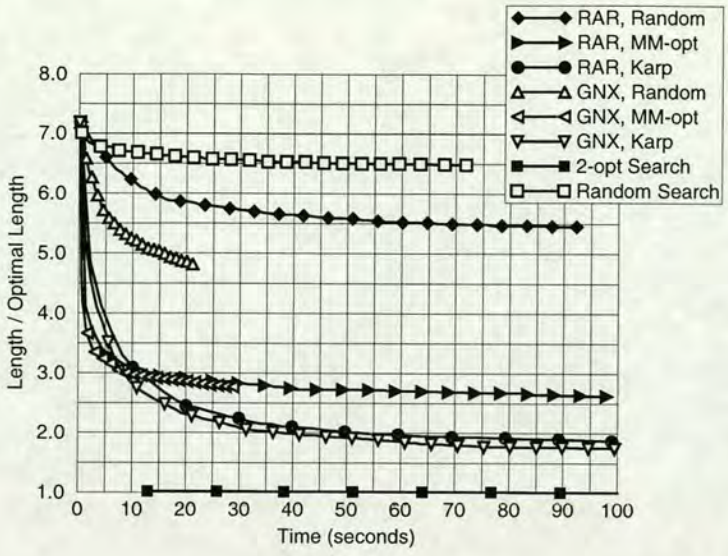


Figure 9.2: The performance of genetic algorithms (i.e. with *no* hill-climbing) on the 100 city Krolak C TSP is shown. The length of the best solution in the population (relative to the length of the optimal tour) is shown as a function of wall-clock time. Error bars are not shown as they are smaller than the tick marks. For comparison, random search (the top line) is shown, as is the performance achieved by repeatedly generating 2-opt solutions (bottom line). Notice that the genetic algorithm is not remotely competitive with random search over 2-opt solutions. Ticks are placed every 5 generations (500 updates) except for the non-adaptive searches. The ticks for random search over 2-opt solutions occur every 100 updates and those for pure random search every 2,500 updates.

ing random and $\hat{\mathcal{M}}$-opt (i.e. minimal-mutation-based) patching, and also using Karp's heuristic[1] which is specific to the TSP, for comparison. In the memetic experiments, rotated cyclic greedy hill-climbing was applied both to the initial population and before children were inserted into the population. Comparisons with random search and with repeated generation of 2-opt solutions are also shown. The final experiment used a larger 442 hole PCB problem from the same test suite, with a reproductive plan based on GENITOR (see chapter 7). The results are shown in figures 9.2–9.5, and are on the basis of "wall-clock" time. In all cases, the length of the best tour in the population is plotted, normalised by the length of the optimum tour. Different algorithms are run for different numbers of updates to give broadly comparable total run times. It should, however, be noted that the implementations of the operators used are not tuned, and the implementation of RPL2 itself was still under beta test at the time of writing, so times should be taken as indicative rather than definitive.

As expected, given the decomposable nature of the evaluation function and the large number of possible alleles for the TSP, the memetic algorithms shown in figure 9.3 all

[1]Karp's heuristic, described in Lawler *et al.* (1985), repeatedly merges together the two longest threads (see chapter 7) in the proto-child until a completed tour results. Each merge is optimal in the sense of exhaustively searching over all potential edge combinations for the join.

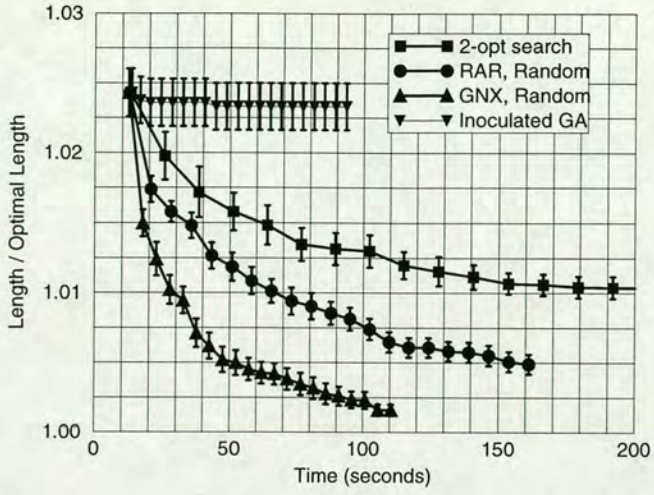


Figure 9.3: The performance of memetic algorithms with random patching on the 100 city Krolak C TSP is shown. The top line shows the performance of the best genetic algorithm (i.e. using G2X and Karp stitching) inoculated with a starting population of randomly generated 2-opt solutions. The second highest line is the same as the bottom line of figure 9.2, i.e. shows the performance of random search over 2-opt solutions, but notice the massively expanded scale on the y-axis. The bottom two lines show that G2X significantly out-performs RAR_2 on this problem. Tick marks are shown every generation (100 updates) except in the case of the inoculated genetic algorithm, where they are every five generations (500 updates), and error bars indicate standard errors.

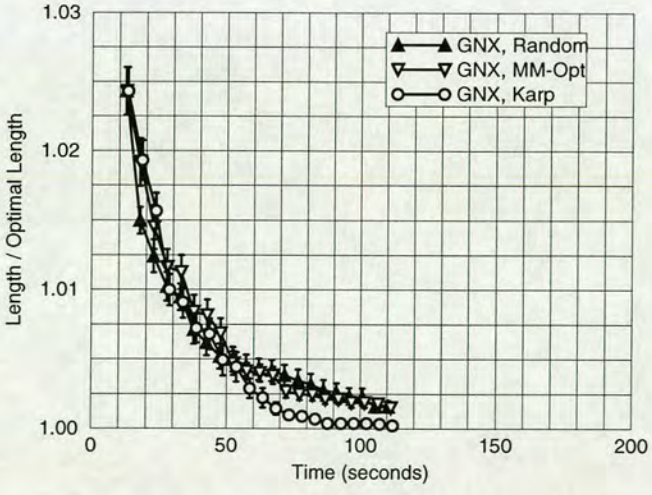


Figure 9.4: The performance variation of memetic algorithms using G2X as a function of patching method is shown for the 100 city Krolak C TSP. Notice that the effect of the patching is rather small, in contrast to the large effect it has on the genetic algorithm (figure 9.2), though Karp stitching still performs best. A similar pattern emerges if RAR is used (not shown), but the performance for each patching method is worse than with G2X. Ticks are shown every generation (100 updates) and error bars indicate standard errors.

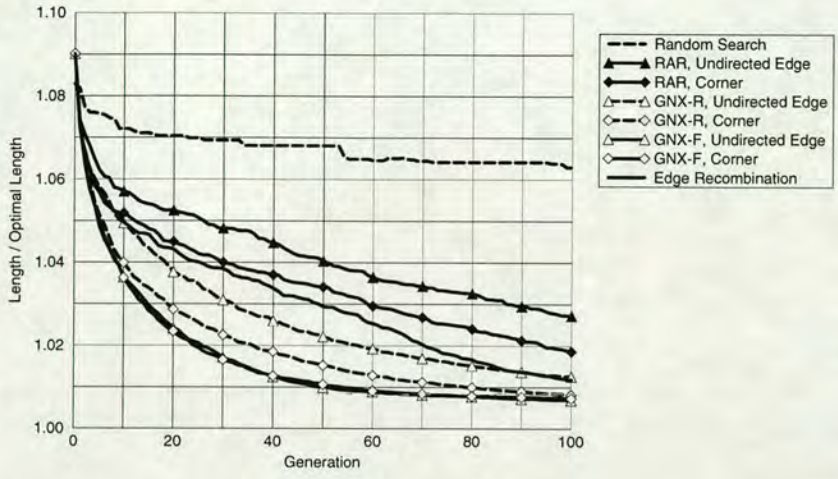


Figure 9.5: The graph shows the same problem as is presented in figures 7.6 and 7.7 (page 43), but now using a memetic algorithm that applies full minimal-mutation-based optimisation before each evaluation. Note the different scales used on the axes. Results for the permutation representation are omitted owing to the extremely long generation times required. Furthermore, even on a per-generation comparison, results are not competitive with those shown. Results are also omitted for directed edges, again because of excessive compute times required. These arise because the minimal mutation for this representation is a 3-change, which requires at least $O(n)$ more computation than the 2-changes needed for undirected edges and corners.

significantly out-performed their genetic counterparts, figure 9.2. (It should be noted that even if full evaluation is performed at each memetic step the overall performance of the memetic algorithms discussed is still superior.) Note particularly that with the exception of the algorithms using Karp stitching, all implementations are direct instantiations of the formal, representation-independent algorithms discussed above.

The choice of patching algorithm has a large effect on the genetic algorithms in figure 9.2, with the $\hat{\mathcal{M}}$-based and Karp stitching providing substantially higher performance, whereas the choice of recombination operator has little effect. Conversely, for memetic search the choice of patching algorithm has relatively little influence over performance (figure 9.4), but here G2X is significantly superior to RAR_2 (figure 9.3). Notice also that genetic algorithms fail by a large margin to match the performance achieved by simply repeatedly generating 2-opt solutions.

The patching results can be understood since in the memetic case the local search will be able to fix any poor patches, whereas this ability is not present in the genetic algorithm. The results for the two recombination operators seem to confirm the finding of chapter 7 that for this problem domain (in the context of the particular reproductive plans chosen) G2X is genuinely superior to RAR_2. The only cases in which this superiority is not exhibited are the genetic runs with good patching, but here the results

suggest that it is the patching that is performing almost all the useful search, masking any distinction between the recombination operators' performances.

Clearly the small problem instance chosen is rather easy for memetic search. (The other Krolak 100 city problems have also been tested, with very similar results.) To emphasize this point further, when a 3-opt-based hill-climber was tested, an initial population of 50 solutions was found to contain two copies of the optimum. The problem was, nevertheless, appropriate for this study given the level of difficulty it provided for genetic search.

9.4 Discussion

This chapter has introduced a representation-independent form of memetic algorithm—a genetic algorithm incorporating local search. By formalising (representation-independent) patching and hill-climbing operators using the same approach as was adopted for recombination and mutation operators, we see how this class of algorithm fits neatly within the general framework developed in chapters 3–6.

When the resulting formal memetic algorithm is instantiated and tested empirically on the travelling sales-rep problem described in chapter 7, we are able to demonstrate its superiority to simple genetic alternatives in this case. A significant increase in performance results from the inclusion of minimal-mutation-based hill-climbing, regardless of the particular recombination or patching strategy employed. (This problem also illustrates how memetic search can be cheaper per evaluation when the fitness function is decomposable—more easily calculated for neighbouring solutions than for random ones—although this is not a large factor in overall performance here.) The results also show that it is the local optimisation that plays the most important rôle in memetic search, with relative differences due to choice of recombination operator or patching strategy much smaller than those observed for otherwise equivalent genetic algorithms.

These results confirm the value of incorporating a directed element to genetic search (local optimisation), but show that this can be accomplished effectively without sacrificing the powerful generality of our representation-independent formalism.

Chapter 10

Initialisation Methods for Incorporating Domain Knowledge

We'll pass the books on to our children, by word of mouth, and let our children wait, in turn, on the other people. A lot will be lost that way, of course. But you can't make people listen. They have to come round in their time, wondering what happened and why the world blew up under them. It can't last.
(Ray Bradbury)

Most techniques for evolutionary search are regarded as weak methods, in that they fail to incorporate domain knowledge explicitly. However, it is recognised (albeit not so widely as might be desirable) that success or failure may depend crucially on the quality of the information implicitly captured by the representation and operators which are employed (e.g. Radcliffe, 1992b). The importance of such prior knowledge of the target function is already widely accepted in the global optimisation community at large (Zhigljavsky, 1991). Nevertheless, much of the research in evolutionary computation takes a problem-independent view. In fact, some authors go so far as to speculate about 'black-box' optimisation (Goldberg, 1989c; Kargupta, 1995), despite the fact that any hopes for strong results here are provably fruitless (Radcliffe, 1992b; Wolpert & Macready, 1995; Radcliffe & Surry, 1995). These authors, amongst others, have argued strongly that useful results (i.e. which distinguish a given method from enumerative search) are only possible when parameterised by characteristics of a problem domain of interest. Ironically, as well as work on black-box optimisation, there have been numerous studies of individual problem *instances*, but disappointingly few on well-specified problem *classes*.

One way in which genetic search often can be strengthened within a given problem domain is through *hybridisation* with other search methods. Davis (1991b) is a particular advocate of this, as are many of the practitioners actively using evolutionary methods in real-world applications. Due to their inherent generality and simplicity,

evolutionary algorithms provide an extremely convenient context in which to embed other techniques, and numerous approaches have been considered.

The addition of specialised move operators incorporating heuristics or exploiting domain knowledge in other ways is often effective (Michalewicz, 1992; Davis & Orvosh, 1993). If a true local-optimisation algorithm exists, these ideas can be extended by applying full local optimisation to each child solution produced before evaluation, yielding a *memetic algorithm* (see chapter 9). Embedding a search technique in the genotype-phenotype mapping is another possibility (Belew *et al.*, 1990; Valenzuela & Jones, 1994). In this case, the evolutionary algorithm may be regarded as searching not directly over the space of solutions, but rather over the space of inputs to the search method with which it is being hybridised.

Although such hybrid approaches allow evolutionary techniques to benefit from existing domain knowledge or heuristics dynamically as the search progresses, additional information may be available in forms which do not readily fit within this framework. In particular, it is often the case that one or more good candidate solutions are known (i.e. solutions which are statistically unlikely to be found in reasonable times by random search). These may be the result of applying a local optimisation algorithm (perhaps one too expensive to embed in an evolutionary search), specialist knowledge or experience (which may be difficult to capture algorithmically), current best-practice, or the result of a previous run. Alternatively, some knowledge of the fitness function (such as partial separability) may be available. A reasonable question is then: in what way (if any) can knowledge be usefully exploited to find better solutions in an evolutionary context?

In section 10.1 we review previous work on the problem of initialisation, highlighting the different approaches taken and the further lines of attack that they suggest. In section 10.2 we use these ideas to explore the theoretical implications of the question posed above, leading to discussion of several practical initialisation strategies. This is followed by an examination of the issues of population diversity and convergence in section 10.3. Several real industrial and commercial applications are used to test the proposed methods in section 10.4 and the results are discussed in section 10.5, which also contains proposals for further work.

10.1 Previous Approaches to Initialisation

Little emphasis is given in the evolutionary computation literature to the problem of initialisation; the near-universal approach is to choose an initial population uniformly at random from the search space (Holland, 1975; Goldberg, 1989c). Particularly in real-world problem domains where hybrid approaches are often adopted for the search itself, it seems likely that more effective strategies must exist. Experience shows that there is a

widespread body of apparently unwritten and unquantified folklore about hybridisation via non-random initialisation. One often hears that non-random initialisation "works" and "has been used successfully for application X" if care is taken to "avoid premature convergence". Specifics, however, are often difficult to pin down.

Grefenstette (1987a) captures the main issues involved in non-random initialisation when he states that:

> ... the initial population ... might be chosen heuristically rather than randomly, with the goal of introducing some helpful building blocks into the gene pool. This should be done carefully since [a genetic algorithm] may quickly converge to a local optimum if the initial population contains a few structures that are far superior to the rest of the population.

He carried out empirical studies of the travelling sales-rep problem using three different initialisation strategies. In the first method he attempted to maximise allelic entropy in the initial population (super-randomness, if you will), while in the other two the initial population contained some fraction of tours generated by the commonly-used nearest-neighbour heuristic and a simple variant thereof. He reported no significant benefit to using heuristic initialisation.

Related work on the travelling sales-rep problem includes results from simulated annealing, such as that of Sourlas (1986). He used conventional simulated annealing to find the (small) set of edges which appeared regularly in tours at some fixed low temperature. He was then able to search for optima much more efficiently using an annealing algorithm which only considered moves involving these *acceptable* edges. He also reported empirical studies which indicated that only short edges appear with high probability in this acceptable set (i.e. each city is very likely to be visited immediately before and after one of its nearest neighbours). This suggests the possibility of initialising based on some non-uniform probability distribution over the set of all edges (which can be viewed as alleles in this problem; Radcliffe & Surry, 1994b). This could obviously be generalised to any domain in which a useful prior distribution over alleles is available, perhaps even developed from previous runs.

The phenomenon of premature convergence (usually taken to mean the situation in which a population has become nearly homogeneous without including a global optimum) is important in evolutionary search. As reviewed below, it has been widely studied, leading to numerous proposed methods for avoiding it, or for restarting search following its occurrence. If one views a single good solution (from whatever source) as the focus of a "converged population", the relevance of such restart methods to the initialisation problem is apparent. Although most previous work in this area has considered only binary representations (sometimes encoding real-valued parameters),

in this chapter we will consider the non-binary representations typically employed in real-world applications (Davis, 1991b). This introduces important technical differences in the character of mass-mutation.

Shaefer (1987) introduced many novel ideas of relevance to initialisation in his 'ARGOT' strategy, which utilised a dynamically adapting representation. Both the resolution and range with which real parameters were represented by binary strings were adapted over time, based on population measures of gene-wise convergence and diversity, helping to avoid premature convergence. Shaefer & Smith (1990) extended ARGOT to handle combinatorial problems such as the travelling sales-rep. In a similar approach, Schraudolph & Belew (1990) considered a dynamic parameter-encoding scheme in which the interval represented by a given gray-coded binary gene shrinks over time, triggered based on measures of population convergence. Both pairs of researchers reported good results on a test suite of problems.

In his CHC genetic algorithm, Eshelman (1990) employed *mass mutation* to restart the search when the population had converged. (In normal operation CHC used no mutation.) During a restart, copies of the best individual found to date were mutated at a high rate (e.g. 35%) to produce the new population. Typically the best individual was also added unchanged. Whitley *et al.* (1991a) instead advocated the *delta-coding* technique for restarts. In this method, chromosomes encoded relative distances from previously discovered good solutions, rather than absolute parameter values. Their algorithm was repeatedly run until convergence and then restarted with the new chromosomes representing displacements from the best previous solution.

It has been reported that even in the absence of known good solutions it can be computationally efficient to allocate more than the minimum resources to initialisation before the search itself takes place. For example, Bramlette (1991) used what he called *extended random initialisation* in which each population member was selected as the best of n randomly chosen individuals. He considered several standard test functions and used a meta-GA to evolve this tournament size (among other parameters), and reported good results with n up to 20, for which about 14% of function evaluations were used in initialisation. The *messy genetic algorithm* of Goldberg *et al.* (1989) used an analogous *primordial phase* to find 'useful' building-blocks.

In real-world applications, the importance of non-random initialisation has often been cited. Fogarty (1989) used a genetic algorithm to optimise valve settings for an industrial furnace application. He experimented with time-varying mutation rates, and found that an effective initialisation strategy involved starting with a completely homogeneous population (cloned from a conservative solution) and using an initially high but exponentially decreasing mutation rate in the ensuing evolutionary process. He also showed that such a mutation schedule was not particularly effective when starting

with a randomised population. There are clear links with the mass-mutation approach but the two methods are not equivalent.

De la Maza (1989) studied the problem of generating production rules to make inferences about a database (here concerning horse-racing). Finding that run-times were excessive when starting from a random population, he instead began with a population consisting of the best single-variable rules produced by a simple heuristic.

Powell *et al.* (1989) combined a genetic algorithm with an expert system for engineering design optimisation (e.g. turbine design). Here, the evaluation function was a complex simulation code with many degrees of freedom. To avoid the perceived inefficiency of evaluating many random solutions, they seeded more than half of the initial population with a combination of previous good solutions and prior design steps made by the expert system. They reported that this reduced by a factor of five the number of runs required to yield a given performance improvement.

Ramsey & Grefenstette (1993) evolved condition-action rules in the face of a changing environment in their SAMUEL system. They employed a restart strategy they termed *case-based initialization*. At the start of each new epoch (environment change) a large part of the population was replaced with a combination of best strategies from similar epochs, robust general strategies, and some random strategies. They claimed that this mix helped to preserve diversity and to avoid premature convergence.

In his work on Tierra, Ray (1994) considered the evolution of computer programs which illustrated many parallels with biological systems, and seeded the initial "primordial soup" with either hand-crafted or previously evolved programs. He introduced the term *inoculation* for the process of non-random initialisation. As defined by the Collins English dictionary,

> **inoculate** *vb.* **1.** to introduce (the causative agent of a disease) into the body in order to induce immunity **2.** to introduce (microorganisms, esp. bacteria) into (a culture medium) **3.** to cause to be influenced or imbued, as with ideas

he used it in its second form. However, it is interesting to note that the third form is particularly apt for the process by which we try to exploit the unknown good characteristics captured by some known high-quality solution(s).

10.2 Initialisation and Inoculation Strategies

We now return to the question of how best to exploit domain knowledge through initialisation. We start by establishing the conditions under which this may be feasible, and go on to suggest various mechanisms by which it might then be accomplished.

10.2.1 Theoretical Considerations

When presented with one or more good solutions to a search problem, but with no additional information, we can argue from fundamental limitations on search that they are useless in the quest for better solutions—that is, the search will perform equally well without them. The so-called "No Free Lunch Theorem" (Wolpert & Macready, 1995) and related results (Radcliffe, 1992b; Radcliffe & Surry, 1995) capture the straightforward idea that in the absence of prior information (i.e. domain knowledge) no search method can outperform enumeration. The essential intuition is that even after sampling an unknown function at some subset of its domain, there is no principled way in which to predict its value at some as-yet unsampled point without making assumptions about the function. Clearly, additional knowledge of the function's structure is required before any particular set of assumptions could be justified. When we consider a set of good solutions and their corresponding function values in the same light, as if they formed some "past search history", we see that they must therefore be useless to further search unless we have some way of making reliable generalisations from the given solutions to unexplored regions of the search domain.

We must therefore posit some additional (domain-specific) knowledge in order to admit the possibility that known good solutions might be useful (and indeed to have any purpose in continuing this discussion). This is not unreasonable as any real problem is likely to contain a great deal of structure. Of course, this does not mean to say that said structure is necessarily easy to exploit!

As discussed previously, evolutionary methods typically make use of domain knowledge only implicitly through the representation and operators that they employ. It thus seems that this is a reasonable mechanism by which to inject the required additional information. That is, we will assume that a context of representation and operators (in some senses duals of one another; Battle & Vose, 1991) is given, and that they are in some loose sense "well-suited" to the problem at hand.

We may then refocus our attention on the problem of how to make use of known good solutions, given also a suitable representation and operators which will be employed in the search. We claim that such high-quality individuals will be useful in finding better solutions only if they share properties with respect to the operators and representation at hand[1]. In the language of evolutionary search, this may mean that the hamming distance between the corresponding genomes is relatively small, or that the average fitness of some schemata (or their generalisation, formae; Radcliffe, 1991a) to which they belong is relatively high. However, because the mechanisms by which

[1]In a somewhat circular fashion, this emphasises that a good first step in constructing a useful representation is to examine explicitly a collection of known good (and perhaps poor) solutions to identify common characteristics that might be used to generate equivalence relations.

evolutionary search "works" are as yet poorly understood, these are by no means the only possibilities.

It should also be noted that the oft-stated goal of finding a global optimum of a given function is poorly conceived unless such an optimum shares characteristics with other high-quality solutions that permit it to be found after visiting only a small fraction of the search space. It is not *in general* the case that a representation which is well-suited to a search problem in the sense discussed above will have the same qualities for the particular points where global optima happen to be located.

The arguments above indicate that any approach to the problem of initialisation should take place within the framework of a specific representation and set of operators, because it is they that determine the "landscape" around the seed solutions. They further suggest that non-random initialisation (using given high-quality solutions) will be most effective if performed using the *same* combination of representation and operators used in the ensuing search—an optima that is "near" an inoculant in one representation may be an isolated outlier in another. This hypothesis could be explicitly tested in the context of any problem for which we can construct multiple natural representations. We proceed to explore some of the possibilities for such initialisation strategies.

10.2.2 Practical Inoculation Strategies

When no heuristic knowledge is available, and no assumptions are made concerning representation or operator quality, we proceed by simply selecting the population uniformly at random from the search space. There is also the option of biasing the sampling to increase the uniformity of coverage of the search space (Holland, 1975; Grefenstette, 1987a).

The simplest strategy for incorporating domain knowledge in the initialisation process is to *inoculate* a random population by adding a good solution and then to let the algorithm take its course. The heuristic will be manipulated by the hypothesised good operators in the well-suited representation, and if it does capture useful information, may help the search to proceed more efficiently. In practice, details of the particular algorithm used will also have dramatic effects on efficacy as discussed in section 10.3.

Given any mutation operator[2], *mass mutation* can be used to inject variants of a good solution into an initial population. In this procedure, a solution is repeatedly cloned and mutated at a high rate to produce each member of the population. This population may also be inoculated with an unmodified copy of the original solution. As the rate of mutation is increased, the resulting population becomes less clustered about the original solution and more like a random one. Note however that with any mutation

[2]One could also envisage using a recombination operator in the initialisation process but this will be pursued in a future paper.

operator that does not chose uniformly between alleles (such as creep mutation) even mass mutation with a rate of 100% does not result in a truly random population.

A further alternative, not requiring a particular good-quality solution, presents itself when allele values are known to capture fitness information in some way (for instance, there may be some degree of linear separability in the fitness function). In this case it is possible to initialise the population with a non-uniform distribution over allele frequencies. This is explored further in the discussion of the travelling sales-rep problem in section 10.4.4.

It is clear that any method for incorporating previously known good-quality solutions into an evolutionary algorithm can also be used for *restarting* the algorithm once it has converged in some sense (see section 10.3). In such a case, the best solution or solutions from the final population can be used to seed a new initial population. However, it is important to distinguish between the two cases. For while we have to posit a good representation and operators in order for an externally provided solution to be potentially useful, in the case of a restart we know that the solutions we are initialising with have been discovered exactly because we used the particular representation and operators that we did. This may appear to be a subtle distinction, but seems to be borne out in experiments—in the case of the TSP (section 10.4.4) we find that we can effectively inoculate with a heuristic tour but that an efficient restart strategy is elusive. Further research is clearly required here.

10.3 Premature Convergence and Population Diversity

A number of practical algorithmic details must be considered when utilising any initialisation procedure. Most important perhaps is the issue of avoiding premature convergence and preserving population diversity, as it is widely believed that inoculation strategies exacerbate these problems. We discuss various ways in which convergence (or conversely, diversity) can be measured, and then mention some methods by which it can be combatted.

Population diversity can be defined in a variety of ways on each of a number of levels. We can consider measuring genotypic diversity, phenotypic diversity, or the diversity of objective-function (fitness) values.

Fitness diversity measures are typically simplest, as they involve only the calculation of parameter estimates to characterise the spread in the distribution of a single real variable (such as variance or higher moments).

Phenotypic diversity measures are normally application dependent, but involve measuring the heterogeneity in expressed characteristics after morphogenesis.

Genotypic measures, at least in the case of genetic or pseudo-genetic representations (Radcliffe & Surry, 1994b), involve measuring the spread of a set of points in a

multi-dimensional vector space (in which the axes correspond to genes and allowable parameter values correspond to alleles). Any distance metric, such as hamming distance (for cardinal genes) or Euclidean distance (for ordinal genes) can be used to collapse the multi-dimensional space into a single-variable distribution which can be analysed using conventional statistical techniques. Alternatively, entropic methods seek to calculate the first-order information content of a typical population member (e.g. Grefenstette, 1987a). They are applicable only in the case when alleles are drawn from a finite set. A further approach would be to employ cluster analysis techniques, although it is not clear whether any work in this area has taken place.

Combined definitions are also possible—for instance Whitley *et al.* (1991a) measure diversity by calculating the hamming distance between the pair of genomes (bit strings in this case) with the best and worst fitnesses.

Numerous techniques have also been developed for countering convergence and thus encouraging diversity. These include exploitation of population structure and speciation (surveyed in Radcliffe & Surry, 1994d), co-evolutionary models (e.g. Hillis, 1991; Husbands & Mill, 1991), adaptive mutation (Whitley & Hanson, 1989), incest-prevention (Eshelman, 1990), crowding (De Jong, 1975) and sharing (Goldberg, 1989c), as well as others. Due to the diverse nature of the problems studied here (section 10.4) we used only simple measures based on the selection and replacement regime. Our algorithms enforced uniqueness within the population and used tournament selection to induce a relatively low selection pressure. These measures help to avoid any difficulties created by the existence of a *super-individual* (a solution substantially better than the rest of the population), which is likely to be the case after inoculation. We also employed elitism in order to prevent the stochastic loss of the best solution (particularly important when only a single inoculant is initially present).

As noted below, we actually find that no unusual loss of diversity is associated with runs using non-random initialisation, but that we do nevertheless often observe convergence to lower-quality solutions than with random initialisation.

10.4 Experimental Results

Quadstone Limited, a decision support company, has worked on a variety of industrial and commercial optimisation projects. Several of these applications provide the basis for experiments on various of the previously discussed initialisation strategies. This brings the twin advantages of having non-trivial, real-world applications rather than artificially constructed problems, and also of building on the extensive prior effort on algorithmic tuning and sensitivity studies.

Four problems are presented, two involving high-dimensional real-valued search domains, and two of a combinatorial nature. They include both constrained and uncon-

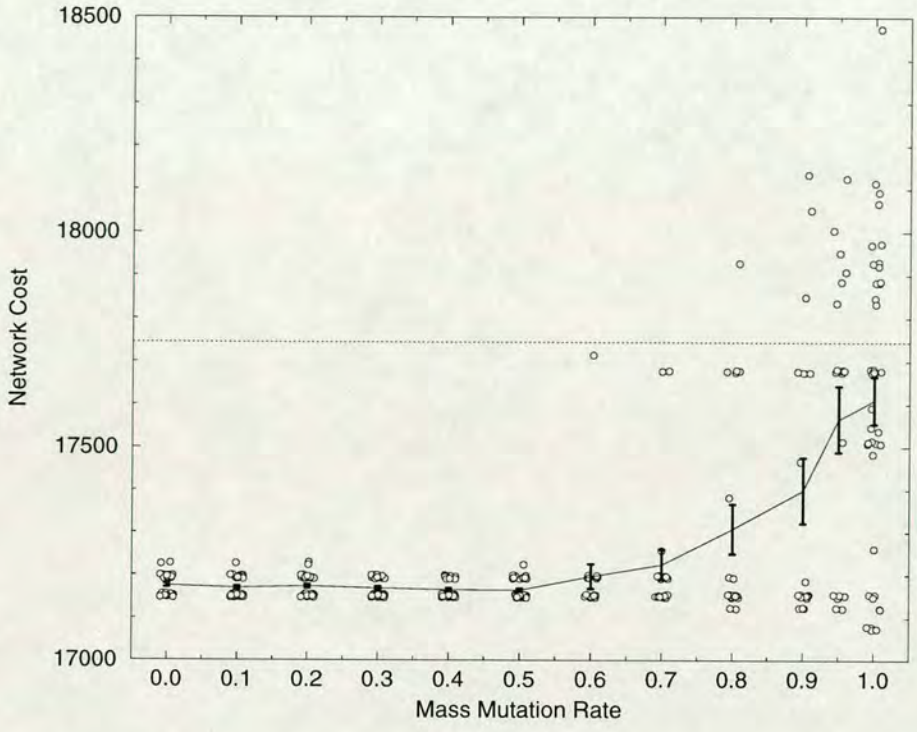


Figure 10.1: Each circle in the figure represents the best (minimum) result found in a single run from an initial population formed by mass mutation of a known good solution (produced by a proprietary heuristic; cost shown as dotted line). A rate of 1.0 corresponds to a completely random population, and a rate of 0.0 is used to indicate a random population containing a single unchanged copy of the inoculant. A small amount of random noise has been added to both the coordinate values in order that multiple identical points can be distinguished. The curve indicates the mean best performance with standard error bars. Although mean performance improves as the population starts more tightly clustered around the heuristic solution (lower mass-mutation rates), the spread also decreases so that the best solution found over a number of runs actually deteriorates. Indeed, the very best networks are found *only* when starting with a random population. See also table 10.1 and figure 10.2.

strained problems. In each case, one or more high-quality solutions is available from external sources. For simplicity we only incorporate a single inoculant in these experiments, though in real applications would clearly exploit all available information. Several initialisation strategies are investigated, including inoculation of a random population with a single instance of a heuristic, mass mutation from clones of a heuristic (but with the heuristic itself not present in the initial population), and random start. Some problem-specific methods are also used.

10.4.1 Gas-network Pipe Sizing

Quadstone staff have worked with British Gas plc on a pipe-sizing problem for a gas-network (Boyd *et al.*, 1994). In this application, a network topology is specified along

Network Cost	Hamming distance from:	
	Heuristic	Best Known
17743.8	(0)	12
17677.3	9	19
17227.4	4	12
17195.3	**5**	**13**
17151.5	**7**	**15**
17125.1	15	5
17075.3	12	(0)

Table 10.1: For solutions commonly discovered by the algorithm, hamming distances between both the heuristic solution (top row) used to seed the populations and from the best known solution (bottom row) are tabulated. (Each solution comprises 25 integer-valued genes, each with six alleles.) The two entries in bold represent the strong attractors shown in figure 10.1, which are closer to the heuristic than the two better solutions. However, it is likely that the topology of the search-space induced by the constraints also plays an important role.

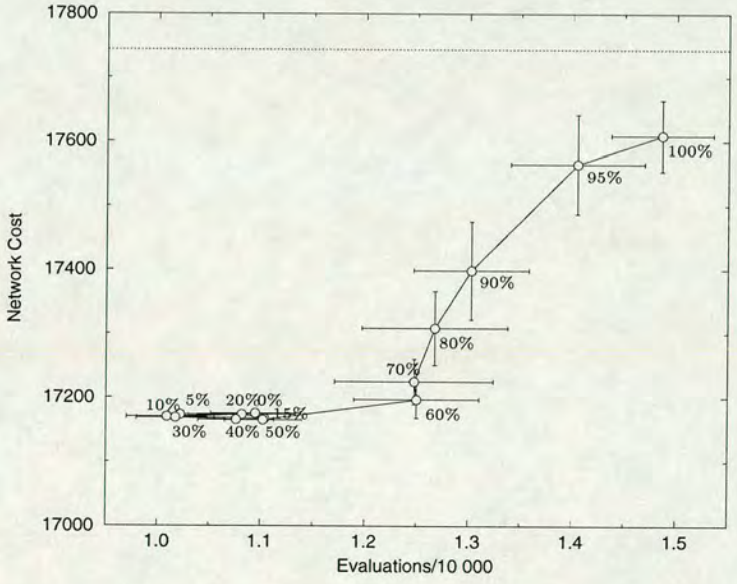


Figure 10.2: This figure summarises the results shown in figure 10.1 (note different scale), but includes time to convergence (defined here as 5000 evaluations without improvement). Each point plots the mean best cost versus mean number of evaluations at convergence for a variety of mass-mutation rates (100% yielding a fully random population, and 0% indicating inoculation with a single unmodified copy of the heuristic). Increasing homogeneity around the heuristic solution (dashed line) tends to increase *average* solution quality and to reduce expected run time, but this hides the fact that best *overall* solution quality over a number of runs actually deteriorates.

with a pattern of nodal gas demands and supplies. The requirement is to select diameters for each pipe segment to achieve lowest cost (smaller pipes being cheaper). The problem difficulty arises due to two constraints—that nodal pressures must exceed a minimum design value, and that each pipe must have at least one upstream pipe of the same or greater diameter. Even a relatively small network (25 pipes) with a small range of allowable diameters (6 for each pipe) leads to a respectably-sized search space ($6^{25} \simeq 3 \times 10^{19}$) in which valid networks are sparse (random sampling indicates that only of the order of 1 in 10^7 networks satisfies both constraints). A network designed by British Gas using a proprietary heuristic had previously been installed, and this provided the high-quality solution used to seed the populations in our experiments.

A previously tuned genetic algorithm based on the COMOGA method (Surry *et al.*, 1995) was used, with populations of 100 networks and terminating after at most 20,000 evaluations or 5,000 with no improvement. Due to the ordinal nature of the integer-valued genes, both creep and random mutation are used, but mass mutation was carried out using random allele replacement. Each of the 600 runs carried out during the experiment converged to a viable network notwithstanding the low density of feasible solutions in the search space. Results are detailed in figures 10.1 and 10.2, along with table 10.1. In general we find that as we incorporate more heuristic knowledge, the average quality of final solutions improves and run-times decrease. However, the variability of final solutions also decreases, and in fact the very best solutions are only discovered when starting with a random population.

10.4.2 Oil-field Production Scheduling

Quadstone has worked with British Petroleum to maximise the economic return from a group of interdependent oil and gas fields (Harding *et al.*, 1998). This involves setting the target production rates for each field in each year. A trade-off arises because earlier extraction offers earlier revenue, but incurs higher costs (since larger facilities are required to produce and handle the flow). There are also more subtle effects, such as the phasing of the start of production for the different fields, and the choice of when to abandon each field. The resulting optimisation problem involves searching for high-quality target production schedules, represented as real-valued matrices with over 200 entries, while satisfying a number of constraints. An existing solution produced by BP using expert knowledge, simulated annealing and sequential quadratic programming was made available to us. In Quadstone's previous work, a hybrid evolutionary technique with specialised operators was employed. A number of representations were considered, using both memetic and genetic approaches, and a large number of parameter sensitivity studies were carried out, resulting in a high-quality, tuned reproductive plan.

We used an algorithm with no local optimisation, population sizes of about 500

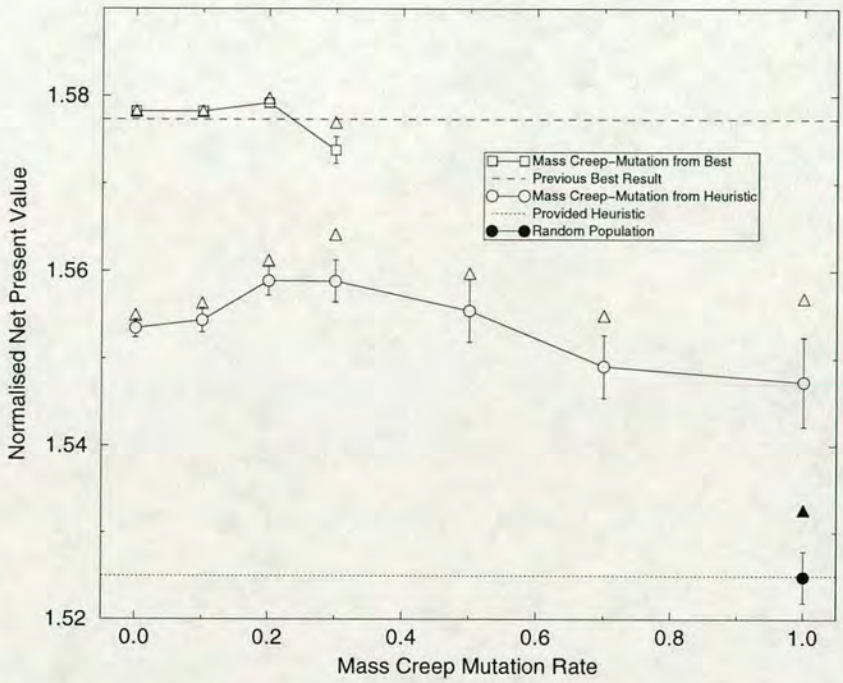


Figure 10.3: The figure shows the results of initialisation experiments on an oil-field production-scheduling problem, involving more than 200 real variables. Two sets of experiments are shown, one using a supplied heuristic solution (dotted line) and one using the best known solution (found using a prior run; dashed line). Mean best performance (maximum net present value) after approximately 250,000 evaluations is plotted with standard errors, and the overall best is shown as a triangle. Mass creep-mutation is used so that even using a rate of 1.0 is not equivalent to starting with a random population (shown as filled symbols). Note the mid-range peaks and decreasing variance with decreasing initial population diversity (lower mass-mutation rates).

solutions and termination after approximately 250,000 evaluations. The results of initialisation experiments using the solution provided and the previously best known solution (found using a memetic algorithm) gave the results shown in figure 10.3. We again find that mean performance improves for intermediate mass-mutation rates but that variability correspondingly deteriorates. Significantly, however, inoculation in this case yields a solution better than any previously discovered.

10.4.3 Credit Scoring

Institutions typically attempt to determine the credit-worthiness of applicants prior to issuing credit facilities. One method for doing this is to use a scorecard to rate an applicant's suitability (e.g. see Hoyland, 1996). The applicant is rated in a number of categories, and these ratings are combined using a set of weights to yield an overall score for the applicant, upon which a decision can be based.

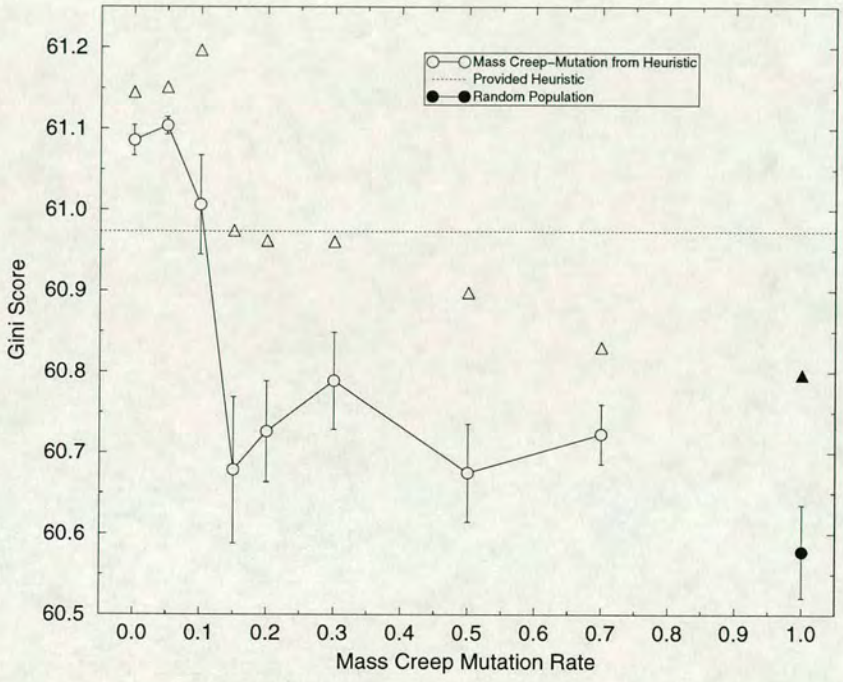


Figure 10.4: The results of inoculating a population of scorecards with an existing heuristic solution (fitness of dotted line) are shown. The mean performance (maximum Gini score) over several runs of the best scorecard found after 10,000 evaluations is plotted, along with with standard error bars and overall best (triangles). Since a creep mutation operator is used in the mass-mutation process, a rate of 1.0 is not equivalent to a random population (shown filled). This figure shows significantly more noise than the other results (figures 10.1, 10.3 and 10.5). This is due in part to averaging over fewer runs which are computationally expensive, but nevertheless similar trends are exhibited: mean performance generally decreases with more random initial populations, but spread increases.

Quadstone has used a variety of proprietary evolutionary algorithms to produce enhanced credit-scoring models. For a linear (additive) scorecard, this involves finding a set of real-valued weights which maximises the predictive power of the scorecard over a database of customers whose credit-worthiness is known. The measure considered here is the Gini score, which indicates how well a proposed scorecard can separate good from bad risks, varying from -100% (perfectly reversed) to 100% (perfect), with random weights leading to scores of about zero.

In the work previously carried out, an existing scorecard was made available. This was used as a heuristic value with which to inoculate the initial population of 100 scorecards. Runs were carried out for 10,000 evaluations (with each requiring the processing and sorting of a database of about 10,000 records); however, greedy techniques are still competitive until significantly more evaluations are permitted. The results, which serve for comparative purposes, are summarised in figure 10.4. Although somewhat noisy, similar trends to the other experiments are apparent.

10.4.4 Travelling Sales-rep

The travelling sales-rep problem, discussed at length in chapter 7, is well-suited to the study of inoculation as there are numerous known heuristics that are effective in finding near-optimal tours. For the purposes of the work here, we used Karp's stitching algorithm which recursively merges sub-tours to form a tour of all cities (a deterministic tour results from stitching the n "sub-tours" generated by considering each city as a single self-loop). The resulting tour was used in the inoculation experiments shown in figure 10.5. A simple genetic algorithm was used with populations of 100 tours and terminating after 50,000 evaluations. (Note that much larger problems can be solved much more efficiently by incorporating local search, but such was not the purpose of the experiments.)

An alternative initialisation procedure was also used, in which tours are not generated randomly, but instead favour short edges. Work is ongoing in this area, but preliminary results are extremely encouraging, particularly when the idea is extended to yield a fast partial local optimisation scheme.

Work on restart strategies for the TSP is also underway, and suggests that inoculation with a heuristic tour may be fundamentally different from restarting using a previously evolved tour. Various mass mutation techniques seem to either have either no or detrimental effects on the convergence properties of the algorithm.

10.5 Discussion

Through experiments on inoculation and mass mutation in a wide range of real-world problem domains, the existing folklore regarding initialisation has been explored. We

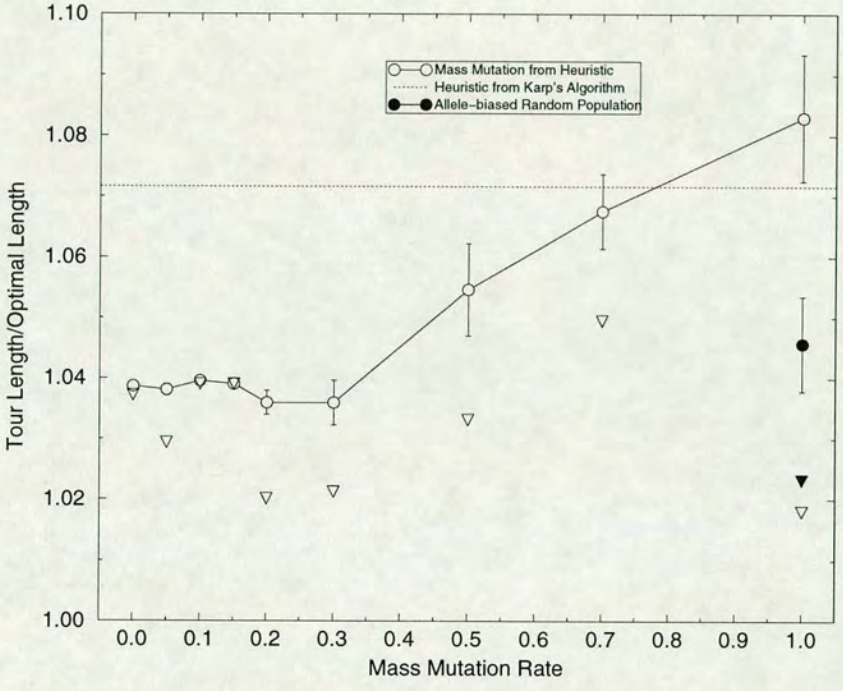


Figure 10.5: The figure summarises the results of initialisation experiments on a small (100 city) travelling sales-rep problem using a simple genetic algorithm with no local search. The tour resulting from (deterministically) Karp-stitching all of the cities together was used as the seed tour (dotted line). The best value found after 100,000 tour evaluations is averaged and plotted along with standard error bars and the overall best (shortest) tour found over a series of runs. The filled points represent a population initialised with tours formed preferentially with short edges (not making use of the heuristic). Note that the best tour found at rate 0.05 can be considered an outlier, as all other runs in that set converged to the same value (near the plotted mean).

find a tendency for mean performance (both in terms of average solution quality and number of evaluations until convergence) to increase as initial populations become more tightly clustered around an existing high-quality solution, peaking at some intermediate level of diversity (corresponding to mass-mutation rates of between 10% to 50%) and then trailing off slightly. However, this increase in mean performance is coupled with a corresponding decrease in variation about that mean, and thus often also with a deterioration in the quality of the best solution found over a number of runs. A particularly striking example of this is found in a pipe-sizing problem, in which the very best networks were only found when starting from a random population (see figure 10.1). Further, there is some indication that when multiple seed solutions are available, best results are obtained when higher-quality solutions are coupled with more tightly clustered initial populations (e.g. lower mass-mutation rates).

Preliminary results also indicate that after the first few generations, there is little quantitative difference in diversity characteristics between algorithms using random initial populations and those using inoculated ones. For instance in both the TSP and pipe-sizing problems (sections 10.4.1 and 10.4.4), we observe little difference in measured first-order entropy (for each problem, the diversity of both inoculated and randomly initialised populations falls to 2–4% of a random population after a few generations). We do however see different convergence patterns, suggesting that diversity is not the sole explanatory factor.

Planned future work includes investigation of the distinction between inoculation with externally provided solutions and restart strategies, recombinative forms of initialisation (as opposed to techniques based on mutation), the importance of representation and the study of fitness distributions near inoculants (using analysis of forma variance and/or fitness-distance correlation measures).

Chapter 11

Constrained Optimisation

> **satisfice** *vb.* *combines "satisfy" and "suffice". Coined in 1956 by the distinguished systems theorist Herbert Simon: "Evidently organisms adapt well enough to 'satisfice'; they do not, in general, 'optimize'." In 1958 he wrote, "To optimise requires processes several orders of magnitude more complex than those required to satisfice."* (Oxford English Dictionary)

It is a frequent criticism of evolutionary algorithms that published results are usually obtained with contrived problems without constraints, leading to the suggestion that evolutionary methods are unsuitable for tackling complex constrained optimisation problems. Within the community, there is a wide-spread perception that penalty function methods are a rather blunt instrument for handling general constraints (e.g. Michalewicz, 1992), exhibiting great sensitivity to the values of their many free parameters, and feeding rather too little information back to the algorithm to allow it to handle the constraints satisfactorily. While other methods are available for problems with explicit constraints (including repair methods, Davis & Orvosh, 1993; smart decoders, Davis, 1987, 1991b; and special operators incorporating problem knowledge, Michalewicz & Janikow, 1991), these do not have fully general applicability, and tend to require significant work for each new class of problems tackled. There is thus a need for a method that combines the generality of penalty function approaches with a greater feedback of information to the underlying search algorithm about the way in which progress is being made with the various constraints under consideration.

In this chapter we present such a method, COMOGA, based on ideas for multi-objective optimisation. Section 11.1 presents an introduction to constrained optimisation and constraint satisfaction problems, and surveys the evolutionary techniques which have been used to tackle such problems. Multi-objective optimisation is reviewed in section 11.2, where the natural fit with population-based algorithms is discussed, and a link is made between multi-criterion optimisation and constraint satisfaction. In section 11.3 these two strands are drawn together to motivate the COMOGA method, which is then described in detail. The technique is demonstrated for a gas-network problem in section 11.4, and results for a test suite of constrained optimisation prob-

lems previously studied by Michalewicz are summarised in section 11.5.

11.1 Constrained Optimisation

11.1.1 Formulation

Many optimisation problems can be phrased as the minimisation[1] of a given function f, over a search domain $\mathcal{S}$:

Minimise

$$f : \mathcal{S} \longrightarrow \mathbb{R}$$

subject to [the solution $x \in \mathcal{S}$ satisfying certain equalities or inequalities].

The equations or inequalities that the solution x must satisfy are known as constraints. Solutions which satisfy all constraints are said to be *feasible* and the set of all such solutions is called the *feasible region*, $\mathcal{S}_F$. (Thus the set of optima, $\mathcal{S}^\star$, is a subset of $\mathcal{S}_F$.) In a *constraint satisfaction problem*, the objective function f is discarded, with the goal being simply to find any solution in $\mathcal{S}_F$.

Constraints can be characterised in various ways. An inequality constraint is said to be *active* at a point x if it is satisfied as an equality at x. It is typical in constrained optimisation that a number of constraints are active for optimal solutions, so that $\mathcal{S}^\star$ is at the boundary of the feasible and infeasible regions (with the result that any weakening of the constraints would change $\mathcal{S}^\star$). Indeed, in highly constrained problems it is often the case that all feasible solutions are near this boundary, and that the volume of the feasible region is negligible when compared with that of the unconstrained search domain. (Although such problems are typically difficult, the converse is clearly not true, with many difficult constrained problems having large feasible regions.)

The constraints on x are conveniently divided into two (imprecise) categories—implicit and explicit. Explicit constraints are those which can be reduced to simple conditions on x and are verifiable "by inspection" while implicit constraints are those which specify a condition on some function of x, that requires significant computation to verify (comparable to, or perhaps much greater than, computing $f(x)$). For example, $x_1 < 3$ and $x_1 + x_2 = x_3$ would normally be regarded as explicit constraints while a condition such as 'the pressure on the wing should not exceed $3\mathrm{N/m}^2$', where the pressure is computed by a fluid-flow simulation, would be regarded as an implicit constraint. The distinction is useful because genetic operators can usually be constructed that respect explicit constraints whereas this is impractical for implicit constraints.

[1] We assume throughout, without loss of generality, that problems are cast as minimisation problems

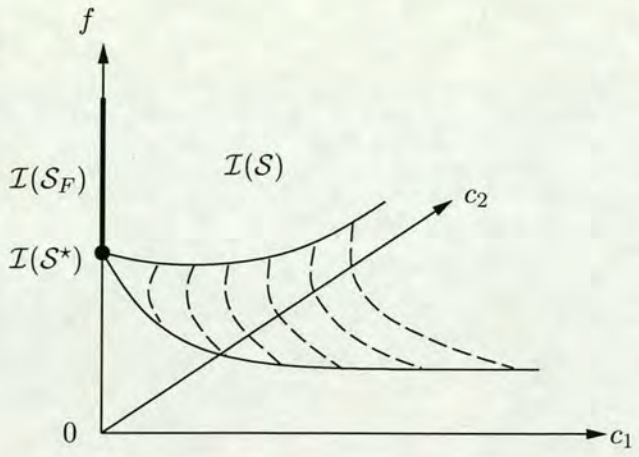


Figure 11.1: A constrained optimisation problem to minimise cost f while satisfying two implicit constraints is illustrated. Points in the search space $\mathcal{S}$ are shown under the three-dimensional mapping $\mathcal{I} \triangleq (c_1, c_2, f)$. The c_1 and c_2 axes measure the degree of constraint violation, so that points in the feasible region, $\mathcal{S}_F$, are mapped to the line where both are zero. The desired set of optima $\mathcal{S}^\star$ is the set of feasible points with minimum cost. All solutions in $\mathcal{S}$ are mapped to points on and above the shaded surface.

11.1.2 Evolutionary Approaches

When a constrained problem is tackled using an evolutionary algorithm, there are several main approaches, with varying degrees of generality (see for example the survey in Michalewicz, 1995b). Perhaps the simplest idea is to restrict the search to the feasible region. This can be done by rejecting infeasible solutions outright, by using greedy decoders or repair mechanisms, or by designing specialised operators which incorporate knowledge of the constraints. The search is thus reformulated as an unconstrained optimisation problem over the reduced space $\mathcal{S}_F$ (the *feasible* region), which is illustrated in figure 11.1. The diagram shows the image of the search space under the vector-valued function $\mathcal{I} : \mathcal{S} \longrightarrow (\mathbb{R}_0^+)^2 \times \mathbb{R}$ where $\mathcal{I} \triangleq (c_1, c_2, f)$ with $c_i(x)$ measuring the degree to which x violates the ith constraint and $f(x)$ giving its cost (to be minimised).

Rejection of infeasible solutions is generally applicable, but is typically limited in its practical utility, due to the low density of feasible solutions in practical problems (e.g. densities of 1 in 10^{10} are not uncommon). In order to avoid generating and rejecting large numbers of infeasible solutions, *greedy decoders* can be used, in which a problem-specific growth function is designed. Here, the genotype does not directly encode a solution in $\mathcal{S}$ but rather a set of parameters which is used by the decoder to generate a feasible solution. Because the decoder must be guaranteed to never produce infeasible solutions (regardless of the provided parameters), it is often extremely difficult to design. Moreover, it is typically hard to generate a decoder that can be guaranteed to be capable of generating optimal or near optimal solutions.

A related approach is to use repair mechanisms to produce feasible solutions from infeasible ones, mapping $\mathcal{S} \longrightarrow \mathcal{S}_F$. (Here, genotypes directly represent solutions in $\mathcal{S}$). This requires a problem-dependent operator which is able to modify any infeasible solution in such a way as to produce a (nearby) feasible solution. Again this is clearly difficult for many types of constraints. When such mechanisms are employed, a further choice is available, namely whether to write the repaired solution back to the genome, or to use it during evaluation, but then to leave the (infeasible) genome intact. The former approach, which is known as Lamarckism, has the advantage of generally allowing faster local improvement, but can make it harder for the search to traverse infeasible regions of the search space, particularly when $\mathcal{S}_F$ is strongly disconnected with respect to the genetic operators. The latter approach, (which has some parallels with the Baldwin effect) has converse advantages and disadvantages. Davis & Orvosh (1993) present anecdotal empirical evidence that writing the repaired solution back to the genome probabilistically, about 5% of the time, is a good option, and the ideas are further explored by Whitley *et al.* (1994).

The final suggestion is to build and use genetic operators that "understand" the constraints, in the sense that the syntactic actions of the operators never produce infeasible solutions (ones that violate the constraints). This approach is advocated by Michalewicz (1992), and Radcliffe (1994). Michalewicz & Janikow (1991) have shown how genetic operators can be built that "understand" linear constraints in this sense, and Schoenauer & Michalewicz (1996) construct operators which maintain solutions on nonlinear analytical constraint surfaces; however, it is clear that for many types of constraints (particularly implicit ones) this approach is impractical.

In a problem with implicit constraints, it is often at least as difficult to determine whether a solution is feasible as to evaluate its cost. In such a case it is typically impossible to utilise repair mechanisms or greedy decoders and impractical to restrict search to simply reject infeasible solutions, and we must take an alternative, generally less attractive approach. By exploring infeasible solutions, the goal is to drive the search towards the feasible region. Particularly in problems with many active constraints at optima, we hope to approach optima from 'both sides', that is to find nearly feasible solutions with better than optimal function values along with feasible solutions with nearly optimal function values (indeed, this is the explicit goal of the segregated genetic algorithm of LeRiche *et al.*, 1995). Thus it has been stated that:

> *[g]ood search should approach the optimal from both sides of the feasible/infeasible border.*
>
> (Richardson *et al.* 1989)

The problem then becomes one of how to compare feasible and infeasible solutions, since in the final analysis it is only feasible solutions which are acceptable. The

most widely applicable approach is to employ a penalty function. Here, the search is treated as an unconstrained problem over $\mathcal{S}$, but the objective function is modified for infeasible solutions by adding terms which degrade their performance. In general, the size of penalty added reflects in some way the *degree* of constraint violation (for example the sum of the constraint violation for each constraint or simply the number of violated constraints). It is also reasonably standard practice (e.g. Richardson *et al.*, 1989; Michalewicz, 1992) to increase the size of penalties during the course of a run (reverse annealing), so that while a degree of violation is tolerated early in early generations, this tolerance reduces over time. This ensures that, after sufficient time, the optimal solutions to the unconstrained problem using the modified objective function coincide with the optimal solutions to the original constrained problem (i.e. $\arg\min_{\mathcal{S}} f_P = \arg\min_{\mathcal{S}_F} f$ for f_P the penalised version of f).

Although penalty functions are essentially universally applicable, they exhibit a number of drawbacks. First, they are weak, in the (formal) sense that they do not provide any problem-specific information to the algorithm. This contrasts with repair mechanisms and problem-specific move operators that exploit understanding of the constraints to provide stronger guidance to the algorithm, but such techniques are not applicable for general constraints. Secondly, the choice of weighting for the constraints is a somewhat subtle matter, particularly when there are many, and increases yet further the number of free parameters to the evolutionary algorithm. Because any choice of parameters determines a fixed form for the modified function in the infeasible region, it induces a fixed ranking on all infeasible solutions. This limits the way in which the search algorithm can explore the infeasible region, since fixed tradeoffs between the various constraints have been introduced. The resulting quality of solution obtained—in fact, the likelihood of finding *any* feasible solution—may be extremely sensitive to the values chosen.

In the next section, we present a method, previously discussed in the context of a specific optimisation problem (Surry *et al.*, 1995), which avoids this difficulty by appealing to the methods of multi-criterion optimisation. Some initial exploration has taken place in this area. For example the work of Schoenauer & Xanthakis (1993) treats each constraint in turn to avoid amalgamating them. Richardson *et al.* (1989) suggest the possibility of using multi-objective techniques (using fitness and either the sum or the number of constraint violations as two objectives) but claim to have been plagued by difficulties. In fact Chu & Beasley (1995) implement a scheme similar to this to deal with a single constraint (using what they term fitness and unfitness). However, they give no guidance in dealing with multiple, non-commensurate constraints, other than by combining them using what is essentially a penalty function (see section 11.2). The method we propose in section 11.3 presents a novel solution to this problem.

11.2 Multi-Objective Optimisation

11.2.1 Formulation

In many real-world optimisation problems there is not a single objective but a set of criteria against which a solution may be measured. Such problems are often known as *multi-objective* or *multi-criterion* optimisation problems, and are defined by a set of objective functions $f_1, f_2, \ldots, f_N$ over the search space $\mathcal{S}$, each of which should ideally be minimised.

Perhaps the most common approach to multi-criterion optimisation is to form a new objective function F that is a weighted sum of the individual objectives,

$$F = \sum_{i=1}^{N} \alpha_i f_i, \quad \alpha_i \in \mathbb{R}^+ \tag{11.1}$$

and to seek to minimise this sum.

If there exists a solution $x^\star \in \mathcal{S}$ that simultaneously succeeds in minimising each of the f_i, this approach can be reasonably satisfactory, because in this case, successful optimisation of F will also optimise each f_i. In the more general case, however, the component objectives f_i will *compete,* in the sense that improvement against one will in some cases require a degradation against another. In this case, the approach of forming a weighted sum is less attractive, because the choice of weights α_i will determine the trade-off between the various component objectives that optima of the combined function F will exhibit. This is particularly unsatisfactory in cases where the various objectives are *non-commensurate,* in the sense that trade-offs between them are either arbitrary or meaningless. A good example of this might arise when seeking to maximise profit while minimising ecological damage, where most people would accept that any assignation of economic cost to ecological damage is at best arbitrary.

In the case of multi-objective problems with competing, non-commensurate criteria, a more satisfactory approach is to search not for a single solution but for that set of solutions that represent the "best possible trade-offs." Such solutions are said to be *Pareto-optimal,* (after Vilfredo Pareto who first advanced the concept) and are characterised by introducing the notion of domination. A solution x is said to *dominate* another solution y if its performance against each of the objective functions is at least as good as that of y, and its performance is better against at least one objective, i.e. if and only if

$$\begin{aligned} & \forall i \in \{1, 2, \ldots, n\} : \quad f_i(x) \leq f_i(y) \\ \text{and} \quad & \exists j \in \{1, 2, \ldots, n\} : \quad f_j(x) < f_j(y). \end{aligned} \tag{11.2}$$

Clearly in this case, x may reasonably be said to be a superior solution to y. The *Pareto-optimal set* (or *front*) $\mathcal{P}$ is the set of solutions that are not dominated by any

other solution in the search space, i.e.

$$\mathcal{P} \triangleq \{x \in \mathcal{S} \mid \nexists\, y \in \mathcal{S} :\ y \text{ dominates } x\}\,. \tag{11.3}$$

11.2.2 Evolutionary Approaches

Although it is difficult to attack multi-criterion problems with traditional optimisation methods, it is relatively natural in population-based search algorithms to consider trying to use the population to hold solutions that represent different trade-offs. Reasonably simple modifications to the selection (and perhaps the replacement) method may be all that is required to effect this. A number of schemes have been proposed, most of which are based around the notion of only allowing selective advantage between solutions when one dominates another. Fonseca & Fleming (1995) provide an overview of many such techniques. The effectiveness of these methods is further enhanced when combined with some form of niching, to encourage greater diversity in the population. Niching methods include structured population models (e.g. Norman, 1988; Manderick & Spiessens, 1989; Gorges-Schleuter, 1989), sharing (Goldberg & Richardson, 1987), and crowding (Cavicchio, 1970; De Jong, 1975).

11.2.3 Constraint Satisfaction as Multi-Criterion Optimisation

It is clear that the constraint satisfaction problem is (formally) equivalent to the simple class of multi-objective problems (discussed above) in which all objectives can be minimised simultaneously. We measure the degree of constraint violation for each constraint (or group of commensurate constraints) and treat each of these as an objective in a multi-criterion problem. (Consider again figure 11.1, where we are now only required to find a solution in $\mathcal{S}_F$. The dashed lines, if extended upwards as vertical manifolds, might indicate a series of progressively dominating surfaces, converging on $\mathcal{I}(\mathcal{S}_F)$—the Pareto-optimal set in this case.) Although, in such a case, minimising a penalty function expressing the degree of constraint violation would be the most common approach, we suggest that a more appropriate strategy when using evolutionary techniques is to use the simple techniques for general multi-criterion optimisation discussed above in order to exploit the ability of the population to hold many different possible trade-offs between the constraints. This allows the algorithm to dynamically "discover" an appropriate trajectory by which to approach the feasible region, rather than arbitrarily assigning the relative importance of different combinations of constraint violations.

In the next section we show how this idea can then be extended to the constrained optimisation problem, where not only must several constraints be satisfied, but a given objective function f must also be minimised.

11.3 The COMOGA Approach

11.3.1 Motivation

It was pointed out in section 11.1 that if we choose to use a penalty function with some given set of parameters to attack a constrained optimisation problem, we make an *a priori* decision about the relative importance of different degrees of constraint violation, regardless of their actual difficulty to satisfy. Further, by combining the degree of constraint violation with the objective function value we impose fixed choice concerning the tradeoff between constraint satisfaction and optimisation (i.e. for every pair of solutions whether feasible or infeasible, their relative attractiveness to the search algorithm is fixed).

In section 11.2, we showed that the methods of evolutionary multi-objective optimisation can be applied directly to the constraint satisfaction problem, and proposed that we could apply similar ideas to constrained optimisation. Obviously, the situation is complicated somewhat by the additional requirement of minimising some function over the feasible region. Here we think of f as an extra criterion which is of less importance than any of the "constraint criteria", i.e. there is no acceptable trade-off between minimising (satisfying) the constraints, and minimising f.

The approach we advocate is to view a constrained optimisation problem alternatively as a constraint satisfaction problem (ignoring the objective function) and as an unconstrained optimisation problem (ignoring the constraints). We further propose to decide adaptively which view to take at any instant based on the relative success with respect to the two formulations. In order to find near-optimal solutions, we must be careful to get neither "too far" from feasibility nor "too far" from optimal fitness, while also recognising that constraint satisfaction is more important than optimisation (as ultimately we are only interested in feasible solutions). We show that an adaptive population-based algorithm is ideal for this purpose.

There are numerous approaches to unconstrained optimisation using a population-based algorithm, and various techniques for constraint-satisfaction based on multi-criterion evolutionary algorithms have been discussed above. The difference between these two types of algorithms can typically be ascribed solely to the selection (and replacement) regime—in the first case selective decisions are based on fitness (cost) while in the second they are normally based on some form of Pareto ranking.

This motivates an attractively simple scheme for coupling the combined constrained optimisation problem, in which we use a single algorithm but randomly decide each time a selective decision must be made whether to consider the problem as a constraint satisfaction problem or as an unconstrained optimisation problem. We then adjust the relative likelihood of adopting each view using a simple feedback mechanism that tries

to maintain a fixed fraction of the population in the feasible region. Because individual solutions can be selected on the basis of either constraint satisfaction or cost, this results in an algorithm which aims to dynamically explore the boundary between feasibility and infeasibility *without* arbitrary penalty factors fixing the relative quality of the different achievable tradeoffs.

This can be seen as a generalisation of the scheme recently proposed by Chu & Beasley (1995) which can importantly handle more than one constraint without having to amalgamate them.

11.3.2 Algorithm

To treat the constraint satisfaction aspect of the problem, we can conceptually label all members of the search space $\mathcal{S}$ with some measure of their Pareto ranking based on constraint violation, either by conceptually peeling off successive non-dominating layers (Goldberg, 1989c), or by assigning to each solution a "rank" equal to the number of solutions which dominate it (Fonseca & Fleming, 1993). (The latter scheme has the advantage that it is easy to subtract the effect of a deleted individual and add the effect of a new individual without re-ranking the entire population.) Note that this ranking is a dynamic one, based on the current population of achievable constraint tradeoffs rather than a fixed ranking of any possible combination of constraint violations. We denote this *population-dependent* ranking function $R : (\mathbb{R}_0^+)^N \longrightarrow \mathbb{Z}^+$, where N is the number of constraints.

From the unconstrained optimisation view, every solution has some cost value associated with it. Thus we are presented with a dual view of each solution in the population and can form the two-dimensional mapping $\mathcal{I}_R : \mathcal{S} \longrightarrow \mathbb{Z}^+ \times \mathbb{R}$, with $\mathcal{I}_R \triangleq (R \circ (c_1, \ldots, c_N), f)$. This reduces the problem to the two-objective problem illustrated in figure 11.2. We must couple the two viewpoints in order to solve the combined constrained optimisation problem: in solving the constraint satisfaction problem we minimise along the R axis and in solving the unconstrained optimisation problem we minimise along the f axis. However, we desire not simply solutions on the Pareto-optimal surface $\mathcal{P}$, but rather solutions in the intersection of the Pareto-optimal set with the feasible region (as constraint satisfaction is "more important" than cost minimisation).

One possible approach is to use a sub-ranking scheme, where only solutions with equal Pareto rank for constraints are distinguished on the basis of cost. However, this is likely to result in an evolutionary process which first concentrates on the constraint satisfaction problem (hence sampling points in the feasible region essentially at random) and only once this is solved tries to reduce cost. This "approach from above" not only lacks the desirable property of being able to combine low-cost, nearly-feasible solutions

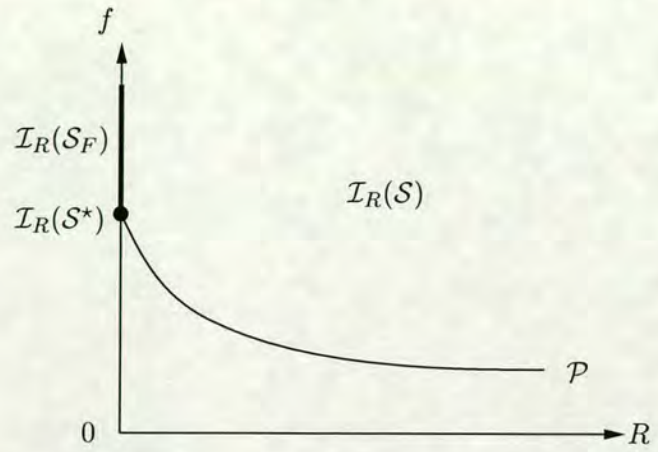


Figure 11.2: The constrained optimisation problem with N constraints can be re-cast as a two-objective problem by assigning a Pareto rank based on constraint violation. The Pareto-optimal set $\mathcal{P}$ is the set of non-dominated solutions under $\mathcal{I}_R \triangleq (R \circ (c_1, \ldots, c_N), f)$. The feasible set is mapped to the line segment $\mathcal{I}_R(\mathcal{S}_F)$, and the desired set of optima is mapped to their intersection, $\mathcal{I}_R(\mathcal{S}^\star)$. The search space $\mathcal{S}$ is mapped to points on and above $\mathcal{P}$.

with higher-cost feasible ones, but may be an extremely poor way to search $\mathcal{S}_F$ if it is a highly sparse and disconnected subset of $\mathcal{S}$.

An appealing alternative approach is to enlist the ideas of Schaffer (1985). In his *vector evaluated genetic algorithm* (VEGA), he selects some fraction (typically $1/k$) of the population based on each of the k objective functions. When a fixed fraction is used for each objective (e.g. $1/k$), this tends to favour the development of "specialist" populations that excel in one objective function, particularly when fitness-proportionate selection is used (Richardson *et al.*, 1989). However, COMOGA will actively exploit this tendency by adaptively changing the likelihood of selecting with respect to each objective.

The suggestion in our case is to use, for example, tournament selection (Goldberg, 1989c), sometimes basing the tournament on cost f and sometimes on the Pareto ranking R with respect to constraint violation. (In cases where the selected attributes are equal, the other attribute is compared.) A probability p_{cost} is used to determine the likelihood of selection with respect to cost, and will be adapted as the algorithm progresses. Any fixed value of p_{cost} will induce an overall probability of reproduction equal to some linear combination of the reproductive probabilities with respect to the two attributes, with *population-dependent* weights. Although such a fixed p_{cost} may favour convergence to some non-feasible point on the Pareto-optimal curve, it is clear that as $p_{cost} \to 0$, the process increasingly favours constraint rank until in the limit of $p_{cost} = 0$ we are essentially solving the constraint-satisfaction problem; seeking feasible solutions regardless of cost (unless the constraint rankings are equal—this is equivalent

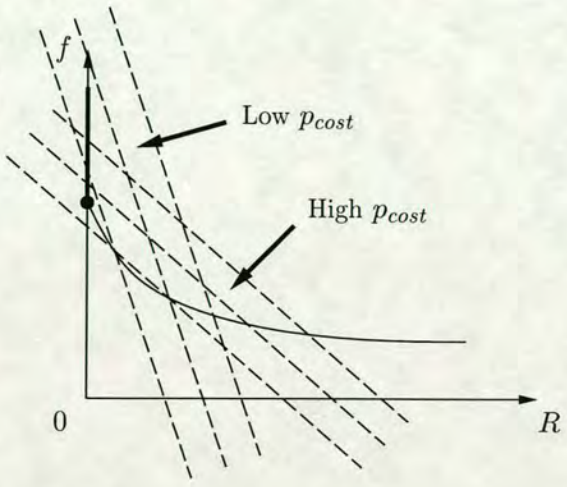


Figure 11.3: Using a VEGA-like scheme of selecting probabilistically with respect to one of the two objectives (cost or constraint rank), we induce a perceived fitness of some population-dependent weighted combination of the two objectives. As p_{cost} tends to zero, the scheme favours constraint rank more, and cost less. By adaptively changing p_{cost} based on the proportion of feasible solutions observed in the population, the algorithm dynamically discovers how to achieve constraint satisfaction and minimisation simultaneously.

to the sub-ranking approach described above). We thus hope that some intermediate non-zero value will allow us to find feasible solutions of low cost. This is illustrated in figure 11.3.

To avoid the problem of fixing a particular value for p_{cost}, we propose to change the value adaptively by setting a target proportion τ of feasible solutions in the population. (τ is similar to the flip threshold of Schoenauer & Xanthakis, 1993.) We start by choosing some arbitrary value for p_{cost}, say 0.5, and some desired proportion of feasible solutions, e.g. $\tau = 10\%$. After each generation, if the number of feasible solutions in the population is not close to τ, then we adjust p_{cost} up or down accordingly: if the actual proportion is too low, we decrease p_{cost}, e.g.

$$p_{cost} \leftarrow (1 - \varepsilon)p_{cost}, \tag{11.4}$$

and conversely, if the proportion is too high, we increase it, e.g.

$$p_{cost} \leftarrow 1 - (1 - p_{cost})(1 - \varepsilon). \tag{11.5}$$

This does, of course, introduce several new parameters to the algorithm (though notably fewer than a penalty function), which we were trying to avoid, but we find in practice that the scheme is remarkably robust to them, in contrast to the sensitivity of penalty function parameters. This leads to the COMOGA method, which is summarised below.

1. Calculate constraint violations $(c_1, \ldots, c_N)$ for all solutions.
2. Pareto rank based on constraint violations (e.g. by counting the number of members of the population dominated by each solution).
3. Evaluate the cost (fitness) of solutions.
4. Select an (expected) proportion p_{cost} of parents based on cost, and the others based on constraint ranking.
5. Apply the genetic move operators (recombination, mutation etc.)
6. Replace an (expected) proportion p_{cost} of solutions based on cost, and the others based on constraint ranking.
7. Adjust p_{cost} if the proportion of feasible solutions in the population is not close to the target proportion, τ, according to equations (11.4) and (11.5). Lowering p_{cost} favours feasible solutions and raising it favours lower cost solutions.

Typical values for the parameters are $\tau = 0.5$ and $\varepsilon = 0.1$, with $p_{cost} = 0.5$ initially.

11.3.3 Summary

The COMOGA scheme has several attractive features. First, and foremost, it removes the necessity for the many parameters of a penalty function which must be determined empirically. Secondly, it turns the acknowledged weaknesses of VEGA to favour extreme solutions to advantage, as in this case we are only ultimately interested in solutions which excel at constraint satisfaction (have low constraint rank). Thirdly, the adaptive approach to specifying p_{cost} allows the algorithm to find its own trajectory to approach the desired optimal values.

In practice, it is important to incorporate explicit constraints (e.g. linear or other specific nonlinear ones) where possible, and to amalgamate multiple commensurate constraints into one or more meta-constraints in order to reduce the dimensionality of the Pareto-optimal front. It may also be advantageous to incorporate niching or other diversity promoting measures in the algorithm (in the work presented here we have simply enforced uniqueness which is not likely to be highly effective for real parameter optimisation).

11.4 An Illustrative Application

We will illustrate the application of the COMOGA approach to a gas-network pipe-sizing problem, contrasting the results with a penalty-function approach. This work has been previously reported in greater detail (Surry *et al.*, 1995).

The problem involves determining the diameters of pipes in a fixed-topology network (with fixed supplies and demands), in order to minimise expenditure, while satisfying two implicit constraints defining the minimal pressure in the network along with an engineering requirement that each pipe should have at least one upstream pipe of the same or greater diameter. (In fact, both constraints could be viewed as meta constraints, each summarising a set of node-level constraints; see Boyd *et al.*, 1994.)

In the particular problem considered, the network contained 25 pipes, each of which could be selected from six possible sizes, giving rise to a search space of size $6^{25} \simeq 3 \times 10^{19}$. The network is a real one, which was actually built using pipe sizes determined by a greedy heuristic method. The density of valid networks (ones which satisfy the constraints) in the search space is low—random sampling of more than 3×10^7 points produced only a single admissible configuration.

Because of the implicit nature of the constraints in the pipe-sizing problem, the only applicable conventional approach is to use a penalty function, as it would be extremely difficult to construct genetic operators that respected them, and prohibitively expensive to use a repair mechanism (if indeed one could be constructed). In order to compare this approach with the COMOGA technique of modifying the selection regime, a standard steady-state elitist duplicate-free evolutionary algorithm using an integer-valued representation and standard recombination and mutation operators was employed (Surry *et al.*, 1995). Binary tournament selection with parameter 1.0 was used to select parents, and the resulting child was re-inserted using a replace-worst scheme. Tournament replacement was also investigated, but the more aggressive replace-worst strategy proved superior.

In the first case, an annealed penalty function which combined a time-dependent weighted sum of the degree of constraint violation for the two constraints along with the basic cost function value was used as the objective. This involved six control parameters to incorporate the two constraints. A wide range of penalty function parameters were tested to discover the typical relative values of constraint violations, etc. As has been widely reported previously, the quality of the resulting algorithms is highly sensitive to these values, with small changes often resulting in runs in which no feasible solution was found.

The technique (with good parameters) produced consistently good results, although it did not always converge to the same solution. In most cases it found networks which were better, often significantly so, than that determined by the heuristic approach. In almost all cases the algorithm found a valid network by the end of the run (i.e. one in which the penalty terms were zero). A snapshot of a single successful run using the penalty function is discussed in figure 11.4.

The same algorithm was then adapted to COMOGA approach, as described in

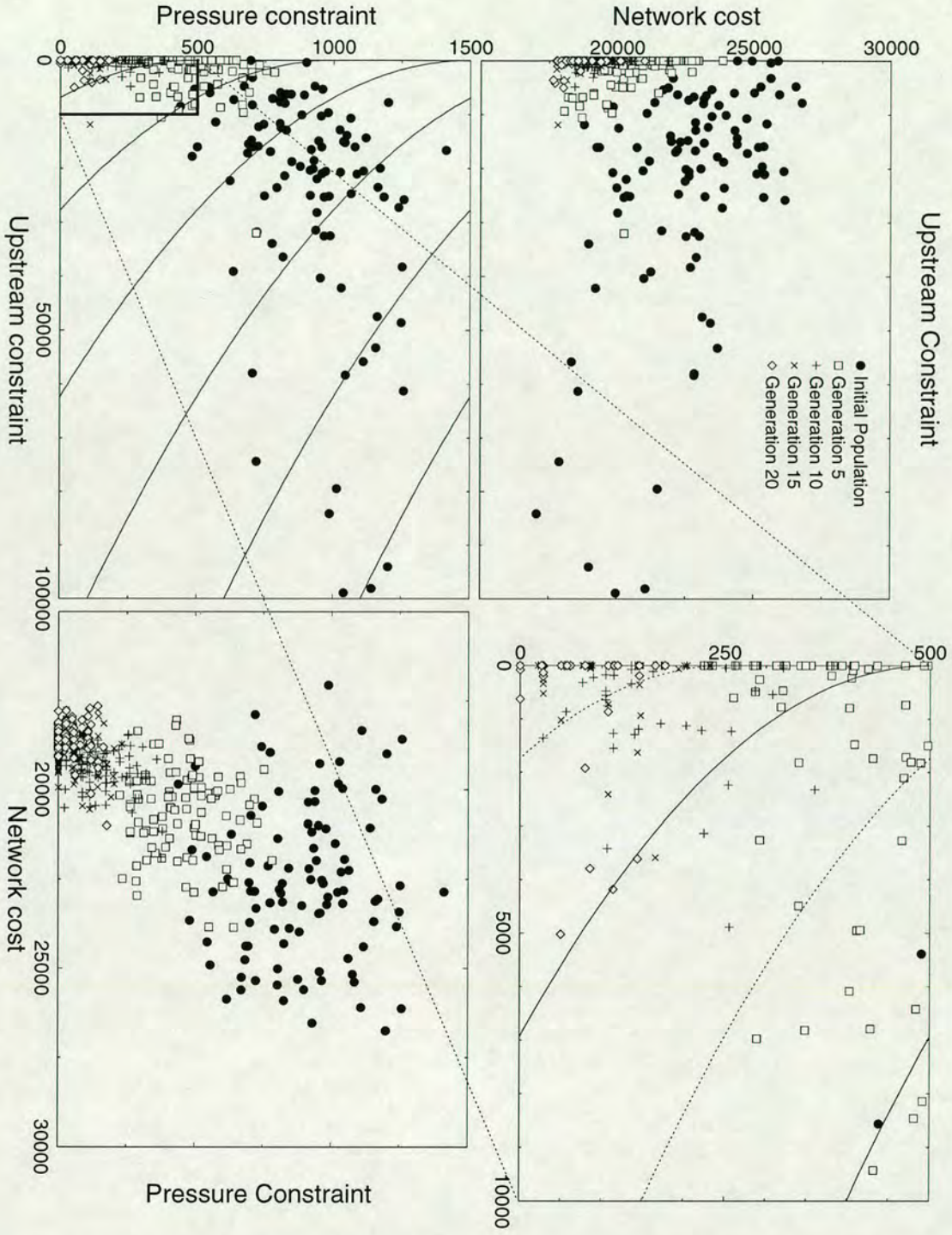


Figure 11.4: The figure shows the pattern of convergence for a single run of an algorithm based on a penalty function. A snapshot of the population is shown every five generations, plotted with respect to the degree of violation of the two constraints and to unpenalised cost (to be minimised). An enlarged section of the region near zero constraint violation is also shown, along with curves which show lines of equal penalty when constraint violations are incorporated with the cost. Note that the population approaches the minimum by first satisfying the constraints and then minimising in the feasible region.

section 11.3. Each member of the initial population was assigned a rank according to constraint violation by counting the number of members in the population by which it was dominated. The same selection and replacement regime was used, but with decisions based on cost value with probability p_{cost}, and otherwise on constraint ranking.

As with the penalty function approach, a variety of population sizes, mutation and crossover rates, and so forth were investigated. Results were best with populations of about 100 individuals, and with the same stopping conditions, runs lasted for similar numbers of evaluations, and produced similar quality solutions (the same "best" solution from the penalty-function approach was found consistently). In contrast to the penalty-function approach however, algorithm performance was much less sensitive to these control parameters. Most importantly, the COMOGA scheme was not particularly sensitive to the method used for adapting the p_{cost} parameter, nor to the target proportion of feasible solutions, τ. A sample run of the COMOGA algorithm is discussed in figure 11.5.

Although the overall performance of the COMOGA algorithm was very similar to that of the best penalty function approach found, both in terms of computational effort required and frequency of finding the best solutions. However, significantly less experimentation was required to find values for COMOGA's parameters that work well than was the case with the penalty function method.

11.5 Experimental Results

In order to validate the COMOGA method it has been applied to a series of known test problems in constrained optimisation for which other evolutionary methods have been applied. The first such problem is the so-called "bump problem" of Keane (1996a), a highly multi-modal maximisation problem defined for an arbitrary number of variables, n, and two non-linear constraints, with an unknown optimum value. For comparative purposes, we studied the problem with $n = 50$. Using an untuned implementation of the COMOGA method, we achieved results superior to any generic evolutionary algorithm previously studied, with fitnesses in the range 0.814 to 0.828 after 200 000 evaluations. Keane (1996b) reports that the best result known to him is 0.832, produced by a non-genetic technique, but more recently Michalewicz & Schoenauer (1996) have achieved 0.833 with a population of size of 30 over 30 000 generations using a problem-specific crossover operator to search the constraint surface, and Wodrich & Bilchev (1997) report results of 0.826 after only 30 000 evaluations, also employing a problem-specific heuristic.

Michalewicz (1995a) has proposed a test set consisting of five constrained optimisation problems, for which he has compared six existing evolutionary techniques. This test set was coded as a library for the *Reproductive Plan Language*, RPL2 (Surry & Rad-

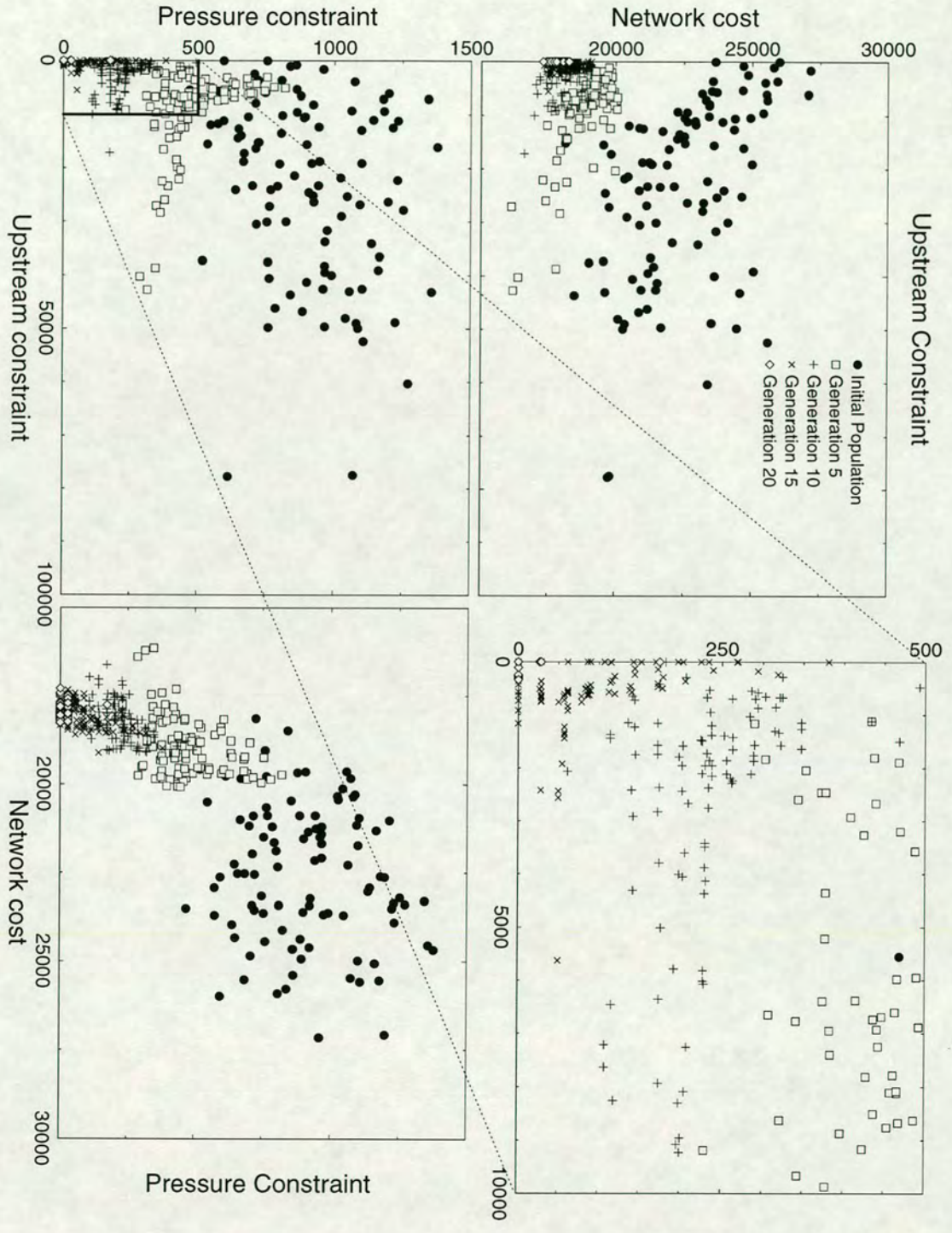


Figure 11.5: The figure shows the pattern of convergence for a single run of the COMOGA method. The population is shown every five generations with respect to the two constraints and the cost. In contrast to the penalty function run, the population explores a variety of tradeoffs between constraint satisfaction and cost, approaches the feasible solutions with minimal cost from both above and below.

Case	Form	Vars	LI	NI	Eq	Optimum	Best/Median/Worst Previous		COMOGA	Rank
1	quadratic	13	9	–	–	−15.000	#4	−15.000	−14.997	5=
								−15.000	−14.996	
								−15.000	−14.994	
2	linear	8	3	3	–	7049.331	#4	7377.976	7081.43	1
								8206.151	7556.85	
								9652.901	8322.51	
3	polynomial	7	–	4	–	680.630	#4	680.642	680.663	1=
								680.718	680.690	
								680.955	680.755	
4	nonlinear	5	–	–	3	0.054	#4	0.054	0.058	3
								0.064	0.205	
								0.557	0.570	
5	quadratic	10	3	5	–	24.306	#2	25.486	24.340	1
								26.905	24.509	
								42.358	24.710	

Table 11.1: The table shows the results of applying the COMOGA method to the test set of Michalewicz (1995a), in which he compared six evolutionary constraint handling methods. The form, number of variables and number and types of constraints (linear inequalities, LI, non-linear inequalities, NI, and non-linear equalities, Eq) are shown for each test problem, along with the optimal value. Results from a set of ten runs are shown for both the previous best method and the COMOGA method, along with the overall ranking on minimum, median and maximum for this new scheme (where this is ambiguous a tie has been awarded). COMOGA performs well on most of the problems, and was the only method to produce a feasible solution in every run for every problem. It performs worst with the linear constraints of problem 1, though this is essentially only a failure to sufficiently polish the parameter values. In problem 4 the equality constraints are treated by essentially converting them to inequality constraints specifying an allowable violation of the equality (a tolerance of 0.001 was used, as this defined feasibility in Michalewicz's study). This creates an artificial division between the feasible and infeasible region.

cliffe, 1994b, Radcliffe & Surry, 1994d), and an untuned implementation of COMOGA was applied to the problems. Table 11.1 summarises the results of these experiments in the same form presented by Michalewicz (based on the same number of total evaluations per run). Here all runs resulted in feasible solutions so Michalewicz's summary by degree of violation has been dropped.

For problem #1, it has failed only to accomplish the final 'polishing' having achieved 4+ figure accuracy in all of the parameter values for every run. This is more suggestive of shortcomings in the genetic operators used (e.g. perhaps the mutation rate or loss of diversity) rather than a failure in the COMOGA technique itself. Note also that no operators which 'understood' these linear constraints were used for the purposes of this initial comparison.

For problem #4, by treating the equality constraints as hard inequality constraints

(using the arbitrary 0.001 factor for allowable degree of violation) we create a sharp artificial boundary between the feasible and infeasible regions. Since this region is essentially a two-dimensional subspace of the five-parameter domain it is perhaps not surprising that it is difficult to "explore" it effectively. In fact, we typically observe that the trajectory of the initial population to find the first feasible solution has a large impact on the quality of the best feasible solution found. This suggests that perhaps this is not the best way in which to incorporate equality constraints within the COMOGA framework, but this requires further investigation.

11.6 Discussion

A new approach of general applicability to constrained optimisation—the COMOGA method—has been presented. This technique treats a constrained optimisation problem sometimes as a constraint satisfaction problem and sometimes an unconstrained optimisation problem, using a single population, by switching between selection regimes. A simple feedback mechanism is used to determine the (expected) proportion of selective decisions which are made with respect to the two viewpoints, depending on the relative progress observed on each.

The COMOGA method uses the memory implicit in the population to "discover for itself" the relative utility of different achievable combinations of constraints and objective function value. The population thus forms not just a pool of good solutions among which recombination takes place, but a context in which to determine the fitness of any one member—the effective weighting of the various constraints is determined by the population, as is the relative weighting of constraint satisfaction and cost minimisation. This contrasts with a penalty function approach, where both are determined *a priori,* and appears to carry the significant benefit of reducing both the sensitivity of the genetic algorithm to the values of the free parameters, and the number of those parameters.

A series of experiments have validated the COMOGA approach, based on the simple yet powerful idea that by merely alternating between two selection (and replacement) regimes we can couple solution of the constraint satisfaction problem to the simultaneous solution of the unconstrained optimisation problem, and discover good solutions to the constrained optimisation problem. Because the scheme is based only on modifying the selection regime, it is possible to use whatever representation, genetic operators and update strategy is appropriate for a given problem within the COMOGA framework.

Chapter 12

Discussion

If you know the enemy and know yourself, you need not fear the result of a hundred battles. (Sun Tzu)

The primary goal of this thesis has been to advocate a much more central rôle to the choice of representation in evolutionary search than it is traditionally afforded. By so doing, we hope to move toward a theory that supports testable predictions about algorithmic performance based on direct measurements of problem characteristics. Although the fundamental ideas explored here are widely applicable to other, more traditional, forms of search, we have focused attention on evolutionary algorithms in particular because such ideas seem most poorly developed there. (For example, alternate forms of the "No Free Lunch" theorem are taken as self-evident within the machine learning community, but are still the subject of ongoing debate amongst practitioners of evolutionary algorithms.)

It is perhaps the case that an attractive biological metaphor and the appealing complexity of the resulting algorithms—arising from selective pressures and biologically-motivated genetic operators acting on a population of solutions represented using a genetic encoding inspired by our own DNA—may obscure the fact that at base an evolutionary algorithm is simply a search algorithm, repeatedly sampling points in a search space based on information inferred from previously visited points. Seen from this viewpoint it is clear that such an algorithm can only be effective if we succeed in incorporating useful problem knowledge in its inductive bias, i.e. the way in which it generalises about problem structure from a limited set of specific observations. By exploring the relevance of the "No Free Lunch" theorem to evolutionary algorithms, both in general and when restricted to polynomial time, we hope further to emphasise this point.

Although it may be tempting to view an evolutionary algorithm as a "black box" technique that exploits a fundamental natural algorithm to perform optimisation (requiring only that we represent our solutions using some simple genetic code), we must not fall into this trap. Indeed the preceding parenthetical remark proves directly our

undoing, for if we do not think carefully about the representational choice we are making, we can guarantee nothing useful about the efficacy of the resulting search. That the need for this decision is often overlooked in problems with a "natural" representation (when the phenotype-genotype distinction is blurred) may explain the lack of attention devoted to representation, but by no means justifies it.

A direct consequence of these investigations is to focus attention on how domain knowledge is captured within the evolutionary framework. Clearly this knowledge derives from the interaction of the genetic coding used to represent solutions and the genetic operators used to manipulate the encoded solutions. We are able to identify three potential mechanisms for capturing problem knowledge in this context. For each new problem we tackle we can:

- design a mapping of solutions to some universal genetic code, and then employ standardised genetic operators to manipulate chromosomes in this canonical representation space;
- design problem-specific genetic operators, conceptually side-stepping the representational issue by acting "directly" on solutions (phenotypes) without employing a distinct genotype; or
- design a problem-specific representation and use predefined operator templates to derive appropriate problem-specific genetic operators.

Of these approaches, we believe that the last is significantly more attractive than the preceding two. The first choice, based on a universal encoding (such as binary strings), is unappealing because problem-specific information is implicitly bound up in the mapping we choose from our solutions to the canonical representation. This means that the genetic operators, acting in the representation space, have only indirect access to such domain knowledge.

The second approach, effectively designing new genetic operators for every problem domain, seems inefficient because it forces us to discard all but the most basic concepts of what defines an evolutionary algorithm. All we can do is draw from a loose collection of biological metaphors to construct a new set of problem-specific operators in each new problem domain. Although this may lead to an efficient algorithm in a particular domain, we are unable to transfer learnings and algorithms between problems.

The third suggestion, advocated here, makes explicit our beliefs about what solution features are important to fitness. By mathematically characterising assertions about problem structure to derive an associated representation, and instantiating template (representation-independent) genetic operators, we are able to construct a problem-specific algorithm that deals directly with these beliefs.

The bulk of the theoretical work in this thesis has explored the implications of this approach. By building on the forma analysis of Radcliffe (1994), we have demonstrated how to characterise mathematically any given problem-domain, thus deriving an explicit representation of that domain that captures our beliefs about the relationship between problem features and fitness. We have introduced a set of well-defined but representation-independent genetic operators that act as templates parameterised by representation. Given any suitable representation we can then formulaically derive relevant, problem-specific recombination and mutation operators.

We have illustrated the ideas in practice by applying the methodology to such diverse domains as combinatorial optimisation (in particular the travelling sales-rep problem) and real-parameter optimisation. In each case we have considered a variety of different representations, based on different characterisations of the problem domain. We have made quantitative measurements of the resulting representations which yield accurate predictions about their relative performance for a fixed search algorithm. Further, we have shown that the techniques work effectively with arbitrary representations. In particular, non-orthogonal representations (in which not all combinations of alleles represent valid solutions) are seen not to be problematical, and indeed we have even demonstrated that mathematically infinite representations can result in computationally finite algorithms that are perfectly feasible to implement. We have also found that the framework is broad enough to encompass related search algorithms such as memetic algorithms (genetic algorithms incorporating local optimisation) and constraint-handling techniques; and motivates ideas for population inoculation and initialisation.

With representation afforded the central rôle it demands, we are left in a position to attack more directly questions about performance prediction, testing explicitly how well any proposed representation captures different kinds of solution structure. Preliminary work reported here links measurements of forma variance to relative algorithmic performance, but this is only the first step towards a testable quantitative theory of performance. By making the choice of representation an explicit part of the design of an evolutionary algorithm we are led directly to numerous obvious avenues for future exploration which, we believe, clarifies the route towards such a theory. Accordingly, it seems fitting to close with what we consider to be key open questions:

- Given a class of problems about which we have partial knowledge, how can that knowledge be formalised in a way useful to constructing representations and algorithms? Ideas from forma analysis offer some directions here, but there is clearly vastly more to be done.

- Can the quality of a representation be measured in any useful way? Forma vari-

ance measurements appear to generate accurate relative performance predictions when an algorithm is fixed, but there may well be more effective approaches.

- What kinds of predictive models of performance (if any) can be built given a well-defined problem domain, representation and algorithm? Currently the predictions based on forma variance are purely qualitative—is there scope for integration with the statistical mechanics approach of Shapiro *et al.* (1994)?

- What performance metrics for algorithms are useful, given the limitations imposed by the "No Free Lunch" theorems?

- How important are revisiting effects in practical algorithms? (Presumably such effects become more pronounced as a search process converges.)

- Can we develop a useful methodology for determining appropriate parameters for a stochastic algorithm, given a representation and problem domain?

- Given an algorithm, can we infer what structure of the fitness landscape induced by the representation and move operators is (implicitly) being exploited by that algorithm—what is the genetic algorithm *really* "processing"? Such insights might allow us to build more powerful models of landscapes (Jones, 1994).

Appendix A

The Reproductive Plan Language: RPL2

The Reproductive Plan Language 2 (RPL2) is an extensible interpreted language for writing and using genetic algorithms and related evolutionary computing paradigms (Surry & Radcliffe, 1994b; Radcliffe & Surry, 1994d). The system was originally developed at the Edinburgh Parallel Computing Centre (EPCC) in collaboration with British Gas plc and Cray Research Inc., and later at Quadstone Limited, an Edinburgh scalable decision support company. It is based on a prototype system, RPL, built under the auspices of the EPCC Summer Scholarship Programme and M.Sc. programme in Computer Science at the University of Edinburgh (Russo (1991)).

In addition to its extensive use throughout this thesis, RPL2 has been applied to a wide variety of real-world applications including retail dealership location (George *et al.*, 1997), stock-market index tracking, pipeline multi-objective optimisation (Surry *et al.*, 1995), the travelling sales-rep problem, neural network topology optimisation, data-mining (Radcliffe & Surry, 1994a) and gas-network pipe sizing (Boyd *et al.*, 1994). The system is now freely available to academic users through a scheme described on the world-wide web (`www.quadstone.com/~rpl2`), and is in use at approximately 20 sites worldwide.

In section A.1, the main features of RPL2 are described, and an illustrative *reproductive plan* is presented. In section A.2, several extensions made to RPL2 during the course of preparing this thesis are described.

A.1 Overview

The principal features of RPL2 are:

- support for arbitrary, user-defined representations
- provision of standard libraries for representation-independent operators (selection, replacement, migration, etc.)

- provision of some standard representations (binary string, variable-cardinality integer string, fixed and variable size sets, permutations, real-parameters, ...) along with associated representation-dependent instantiations of various formal genetic operators
- extensible interpreted language for easy experimentation and manipulation of reproductive plans
- modularity: new representations and operators can be added in a coherent manner, allowing user-customisation of the system
- support for structured population models (island model, diffusion/cellular model) as well as hybrid models and unstructured (panmictic) populations
- portability across a wide variety of serial and distributed-memory MIMD parallel computers
- automated platform-independent parallelism using both data decomposition and task-farming as appropriate

The system is described more fully in the RPL2 manual (Surry & Radcliffe, 1994a), which includes an extensive glossary of terms in evolutionary computation. For illustrative purposes, a simple reproductive plan is shown overleaf.

```
% This is a simple panmictic (unstructured) example that illustrates how
% parallelism can be applied to such problems (the forall construct)
% The plan is based on a population/cache model with generational update.

plan(PanmicticExample)

use     StdInst, Binary(128);           % parameterised by string length
string  sFile;
bool    bMaxIsBest;

int     iCounter, nGenerations, i, nPopsize;

genome  gNew,gChild,gParentA,gParentB;
gstack  gsPop, gsCache, gsParents;      % population, cache, and parents

sFile := "stdout";                      % direct output to the terminal
nPopsize := 200;                        % population size
nGenerations := 100;                    % number of generations
bMaxIsBest := TRUE;                     % maximise the evaluation function

Randomize(0);                           % pseudo-random initialisation

for iCounter := 1 to nPopsize           % create initial population
        gNew := RandomGenome();
        Push(gNew,gsPop);
endfor

forall gChild in gsPop                  % parallel evaluation of population
        EvalOneCount(gNew);             % number of 1s in string
endforall

for i := 1 to nGenerations              % generational update scheme

        Empty(gsCache);                 % empty the stack for next generation
        Empty(gsParents);               % empty the array of parents

        ScaleRanked(gsPop, bMaxIsBest, 0.0, 1.0);       % scale on ranking
        SelectScaledSUS(gsPop, 2 * nPopsize, gsParents); % choose all parents

        for iCounter := 1 to nPopsize   % create new generation of genomes
                gParentA := Pop(gsParents);  % do 3pt crossover,
                gParentB := Pop(gsParents);  % with 20% clone rate
                gChild := CrossNpt(gParentA, gParentB, 3, 0.8);
                Push(gChild,gsCache);
        endfor

        forall gChild in gsCache        % mutate and evaluate in parallel
                Mutate(gChild, 0.01);   % bit-wise mutation rate of 1%
                EvalOneCount(gChild);
        endforall

        Swap(gsPop, gsCache);           % swap the new generation for the old

        StatsPrint(i,10,gsPop,sFile);   % collect population statistics
endfor
endplan
```

A.2 Extensions

Various facilities were added to RPL2 in the course of preparing this thesis. These have not yet been made available in the publically distributed version of the software. The primary areas of work consist of added support for multi-objective optimisation, along with support for compound genomes. In addition, numerous new representations and operators were implemented, but will not be described in detail here.

A.2.1 Multi-objective Optimisation

For the work described in chapter 11, RPL2 was extended to support the notion of genomes with multiple fitnesses. This required changes to the underlying framework, as well as provision at user level of new representation-independent genetic operators to allow manipulation of multiple fitnesses. A number of representation-independent ranking, selection and replacement operators were also defined to support experimentation with different forms of multi-objective optimisation.

Extensions to the RPL2 programmer's framework permitted user-defined evaluation functions to return multiple fitness values for a single genome:

```
void    GUseMultiFitness(GENOME *pg, int n);
double* GMultiFitnesses (GENOME *pg);
double  GMultiFitness   (GENOME *pg, int i);
void    GSetMultiFitness(GENOME *pg, int i, double rFitness);
int     GNMultiFitnesses(GENOME *pg);
```

The `Pareto` library was also developed to provide RPL2 operators which could sensibly manipulate the resulting genomes, for instance using multiple fitness values to do Pareto ranking and generate a single scaled fitness value:

```
% Pareto-rank a population of genomes
ScalePareto(gstack gs, int rankMethod);

% Update an existing Pareto ranking after replacing one genome
ScaleParetoUpdate(gstack gs, genome in, genome out, int rankMethod);

% Compare two genomes with multiple fitnesses using Pareto dominance
% Return -1 if A dominates B, +1 if B dominates A, and 0 otherwise
int ParetoCompare(genome A, genome B);

% Select from a population using a COMOGA-style tournament
SelectCOMOGATournament(gstack gs, bool maxIsBest, int tournSize,
        real probOfBest, bool withReplacement, real probRawFitness);

% Replace within a population using a COMOGA-style tournament
ReplaceCOMOGATournament(gstack gs, genome new, bool maxIsBest,
        int tournSize, real probOfWorst, bool withReplacement,
        bool elitist, real probRawFitness);
```

A.2.2 Compound Genomes

Although RPL2 supports a growing (and user-extensible) variety of representations, it appeared in a number of cases that an appropriate representation for a given problem could be best expressed as a combination of two (or more) of the existing representations. For instance, a subset-selection problem with ordering might be represented using a combination of one of the existing set representations and the permutation representation. To support this, the framework was extended to allow the definition of compound genomes defined using any combination of existing RPL2 types or representations.

For example, in an evolution-strategy algorithm we might require genomes that contain both an actual real-parameter vector, along with a vector of mutation rates for each parameter. Using compound genomes, this could be accomplished easily:

```
...

% Declare a template compound genome structure

compound ES
        genome  coords;
        genome  sigmas;
        real    mutationRate;
endcompound

% Declare an instance of the template

genome  child;

% Initialise the components of the child

child.coords := Real::RandomGenome();
child.sigmas := Real::RandomGenome();
child.mutationRate := RandReal();

...
```

Because the compound structure is itself a genome, it can be assigned fitness values like any other genome, and can thus be used with all of the representation-independent operators already defined within RPL2.

Bibliography

D. H. Ackley, 1987. *A connectionist machine for genetic hillclimbing.* Kluwer Academic Press, Boston.

J. Antonisse, 1989. A new interpretation of schema notation that overturns the binary coding constraint. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

T. Bäck and H.-P. Schwefel, 1993. An overview of evolutionary algorithms for parameter optimisation. *Evolutionary Computation*, 1(1):1–24.

T. Bäck, F. Hoffmeister, and H.-P. Schwefel, 1991. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann (San Mateo).

T. Bäck, U. Hammel, and H.-P. Schwefel, 1997. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):1–15.

J. E. Baker, 1987. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale).

D. L. Battle and M. D. Vose, 1991. Isomorphisms of genetic algorithms. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 242–251. Morgan Kaufmann (San Mateo).

R. K. Belew and L. B. Booker, editors, 1991. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo, CA).

R. K. Belew, J. McInerny, and N. N. Schraudolph, 1990. Evolving networks: Using the genetic algorithm with connectionist learning. Technical Report CS90–174, UCSD (La Jolla).

R. Belew, 1992. Paradigmatic over-fitting. GA-List v6n18, 29th May.

A. D. Bethke, 1980. *Genetic Algorithms and Function Optimizers.* PhD thesis, Univerity of Michigan.

C. Bierwirth, D. C. Mattfeld, and H. Kopfer, 1996. On permutation representations for scheduling problems. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *Parallel Problem Solving from Nature IV*, pages 310–318. (Springer-Verlag, LNCS 1141).

G. Bilchev and I. C. Parmee, 1996. Learning the "next" dimension. In T. C. Fogarty, editor, *AISB96 Workshop on Evolutionary Computation.* Springer ??Update??

L. Booker, 1987. Improving search in genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing.* Pitman (London).

M. J. Box, D. Davies, and W. H. Swann, 1969. *Non-linear Optimization Techniques.* Oliver and Boyd Ltd.

I. D. Boyd, P. D. Surry, and N. J. Radcliffe, 1994. Constrained gas network pipe sizing with genetic algorithms. Technical Report EPCC–TR94–11, Edinburgh Parallel Computing Centre.

M. F. Bramlette, 1991. Initialization, mutation and selection methods in genetic algorithms for function optimization. In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

C. Bridges and D. E. Goldberg, 1987. An analysis of reproduction and crossover in a binary-coded genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale, New Jersey).

R. A. Caruana and J. D. Schaffer, 1988. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proceedings of the 5th International Conference on Machine Learning.* Morgan Kaufmann (Los Altos).

D. J. Cavicchio, 1970. *Adaptive Search Using Simulated Evolution.* PhD thesis, Univerity of Michigan.

P. C. Chu and J. E. Beasley, 1995. A genetic algorithm for the set partitioning problem. Technical report, The Management School, Imperial College. (Revised 1997).

J. C. Culberson, 1996. On the futility of blind search. Technical Report TR 96–18, University of Alberta.

R. Das and D. Whitley, 1991. The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 166–173. Morgan Kaufmann (San Mateo).

Y. Davidor, 1990. Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems*, 4:369–383.

L. Davis and D. Orvosh, 1993. Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 50. Morgan Kaufmann (San Mateo).

T. E. Davis and J. C. Principe, 1993. A markov chain framework for the simple genetic algorithm. *Evolutionary Computing*, 1(3):269–288.

L. Davis, 1987. *Genetic Algorithms and Simulated Annealing.* Pitman (London).

L. Davis, 1991a. Bit-climbing, representational bias, and test suite design. In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

L. Davis, 1991b. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold (New York).

R. Dawkins, 1976. *The Selfish Gene.* Oxford University Press (Oxford).

K. A. De Jong, 1975. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan.

M. de la Maza, 1989. A SEAGUL visits the race track. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

A. E. Eiben, E. H. L. Aarts, and K. M. V. Hee, 1990. Global convergence of genetic algorithms: A markov chain analysis. In H. P. Schwefel and R. Manner, editors, *Parallel Problem Solving From Nature*, pages 4–12. Springer-Verlag.

T. M. English, 1997. Information is conserved in optimization. Technical Report ?, Computer Science Department, Texas Tech University.

L. J. Eshelman and D. J. Schaffer, 1992. Real-coded genetic algorithms and interval schemata. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann (San Mateo, CA).

L. J. Eshelman, 1990. The CHC adaptive search algorithm: How to have safe search when engaging in non-traditional recombination. In *Foundations of Genetic Algorithms.* Morgan Kaufmann (San Mateo).

L. J. Eshelman, editor, 1995. *Proceedings of the Sixth International Conference on Genetic Algorithms.* Morgan Kaufmann (San Francisco).

E. Falkenauer, 1994. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144.

T. C. Fogarty, 1989. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

L. J. Fogel, A. J. Owens, and M. J. Walsh, 1966. *Artificial Intelligence Through Simulated Evolution.* Wiley Publishing (New York).

C. M. Fonseca and P. J. Fleming, 1993. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423. Morgan Kaufmann (San Mateo).

C. M. Fonseca and P. J. Fleming, 1995. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16.

S. Forrest, editor, 1993. *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo, CA).

B. R. Fox and M. B. McMahon, 1991. Genetic operators for sequencing problems. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms.* Morgan Kaufmann (San Mateo).

B. Freisleben and P. Merz, 1996. New genetic local search operators for the traveling salesman problem. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *Parallel Problem Solving from Nature IV*, pages 890–899. (Springer-Verlag, LNCS 1141). impressive results on large (n¿1000) TSP instances.

M. R. Garey and D. S. Johnson, 1979. *Computers and Intractibility: A Guide to the theory of NP-completeness.* W.H. Freeman and Company.

F. A. W. George, N. J. Radcliffe, M. A. Smith, M. Birkin, and M. Clarke, 1997. Spatial interaction model optimisation on parallel computers. *Concurrency: Practice and Experience*, 9(8).

F. Glover, 1986. Future paths for integer programming and links to artificial-intelligence. *Computers and Operations Research*, 13(5):533–549.

F. Glover, 1995. Tabu search fundamentals and uses. Technical report, Graduate School of Business, University of Colorado.

D. E. Goldberg and R. Lingle Jr, 1985. Alleles, loci and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale).

D. E. Goldberg and J. Richardson, 1987. Genetic algorithms for multimodal function optimisation. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum Associates (Hillsdale).

D. E. Goldberg, B. Korb, and K. Deb, 1989. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530.

D. E. Goldberg, 1989a. Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems*, 3:129–152.

D. E. Goldberg, 1989b. Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems*, 3:153–171.

D. E. Goldberg, 1989c. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley (Reading, Mass).

D. E. Goldberg, 1990. Real-coded genetic algorithms, virtual alphabets, and blocking. Technical Report IlliGAL Report No. 90001, Department of General Engineering, University of Illinois at Urbana-Champaign.

M. Gorges-Schleuter, 1989. ASPARAGOS: an asynchronous parallel genetic optimization strategy. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann (San Mateo).

J. J. Grefenstette and J. E. Baker, 1989. How genetic algorithms work: A critical look at intrinsic parallelism. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, 1985. Genetic algorithms for the travelling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale).

J. J. Grefenstette, 1984. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165.

J. J. Grefenstette, editor, 1985. *Proceedings of an International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale).

J. J. Grefenstette, 1987a. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing.* Pitman (London).

J. J. Grefenstette, editor, 1987b. *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale).

J. J. Grefenstette, 1992. Deception considered harmful. In *Foundations of Genetic Algorithms 2*, pages 75–91. Morgan Kaufmann (San Mateo, CA).

J. Grefenstette, 1995. Virtual genetic algorithms: First results. Technical report, Naval Research Laboratory.

B. Groβ, U. Hammel, P. Maldaner, A. Meyer, P. Roosen, and M. Schütz, 1996. Optimization of heat exchanger networks by means of evolution strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *Parallel Problem Solving from Nature IV*, pages 1002–1011. (Springer-Verlag, LNCS 1141).

T. J. Harding, N. J. Radcliffe, and P. R. King, 1998. Hydrocarbon production scheduling with genetic algorithms. *SPE Journal*, pages 99–108.

G. R. Harik and D. E. Goldberg, 1996. Learning linkage. In R. K. Belew and M. D. Vose, editors, *Foundations of Genetic Algorithms IV.* Morgan Kaufmann (San Mateo, CA).

W. Hart, T. Kammeyer, and R. Belew, 1994. The role of development in genetic algorithms. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 315–332. Morgan Kaufmann (San Francisco, CA).

W. D. Hillis, 1991. Co-evolving parasites improve simulated evolution as an optimization procedure. In S. Forrest, editor, *Emergent Computation.* MIT Press (Cambridge, MA).

R. Hofmann, 1993. Examinations on the algebra of genetic algorithms. Diploma Thesis, Technical University of Munich, Department of Computer Science.

J. H. Holland, 1975. *Adaptation in Natural and Artificial Systems.* University of Michigan Press (Ann Arbor).

J. Horn, D. E. Goldberg, and K. Deb, 1994. Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature III.* Springer Verlag, LNCS 866, Berlin.

C. Hoyland, 1996. *Data-driven decisions for consumer lending: credit scoring techniques for risk management.* Lafferty Publications.

P. Husbands and F. Mill, 1991. Simulated co-evolution as the mechanism for emergent planning and scheduling. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

T. Jones and S. Forrest, 1995. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. Technical Report 95–02–022, Santa Fe Institute.

T. Jones, 1994. A model of landscapes. Technical Report n/a, Santa Fe Institute.

T. C. Jones, 1995. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico.

H. Kargupta, 1995. *SEARCH, polynomial complexity, and the fast messy genetic algorithm*. PhD thesis, Univerity of Illinois at Urbana-Champaign. IlliGAL Report No 95008.

S. A. Kauffman, 1993. *The origins of order: self-organization and selection in evolution*. Oxford University Press (New York).

A. J. Keane, 1996a. *Modern Heuristic Search Methods*, chapter A Brief Comparison of Some Evolutionary Optimization Methods, pages 255–272. J. Wiley.

A. J. Keane, 1996b. Personal communication. (unpublished).

J. Kingdon and L. Dekker, 1995. The shape of space. Technical Report ??, University College London.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, 1983. Optimisation by simulated annealing. *Science*, 220(4598):671–680.

J. R. Koza, 1990. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Stanford University.

J. R. Koza, 1991. Evolving a computer to generate random numbers using the genetic programming paradigm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 37–44. Morgan Kaufmann (San Mateo).

J. R. Koza, 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford Books, MIT Press (Cambridge, Mass).

J. R. Koza, 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press (Cambridge, Mass).

E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, 1985. *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimisation*. Wiley.

R. G. LeRiche, C. Knopf-Lenoir, and R. T. Haftka, 1995. A segregated genetic algorithm for constrained structural optimization. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 558–565. Morgan Kaufmann (San Francisco).

G. Liepins and M. Vose, 1990. Representational issues in genetic optimization. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:101–115.

S. J. Louis and G. J. E. Rawlins, 1993. Pareto optimality, ga-easiness and deception. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo, CA).

W. G. Macready and D. H. Wolpert, 1996. On 2-armed gaussian bandits and optimization. Technical Report SFI-TR-96-???, Santa Fe Institute.

B. Manderick and P. Spiessens, 1989. Fine-grained parallel genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 428–433, San Mateo. Morgan Kaufmann Publishers.

A. J. Mason, 1991. Partition coefficients, static deception and deceptive problems for non-binary alphabets. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 210–214. Morgan Kaufmann (San Mateo).

K. I. M. McKinnon, 1972a. Evolutionary processes and machine learning. (unpublished research proposal, Peterhouse, Cambridge).

Z. Michalewicz and C. Z. Janikow, 1991. Handling constraints in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157. Morgan Kaufmann (San Mateo).

Z. Michalewicz and M. Schoenauer, 1996. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32.

Z. Michalewicz, 1992. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag (Berlin).

Z. Michalewicz, 1995a. Genetic algorithms, numerical optimization, and constraints. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 151–158. Morgan Kaufmann (San Francisco).

Z. Michalewicz, 1995b. A survey of constraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155. MIT Press (Cambridge, MA).

T. M. Mitchell, 1980. The need for biases in learning generalizations. Technical Report CBM-TR-117, Computer Science, Rutgers University.

T. M. Mitchell, 1997. *Machine Learning.* McGraw-Hill.

D. J. Montana and L. Davis, 1989. Training feedforward neural networks using genetic algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767.

P. Moscato and M. G. Norman, 1992. A "memetic" approach for the travelling salesman problem — implementation of a computational ecology for combinatorial optimisation on message-passing systems. In *Proceedings of the International Conference on Parallel Computing and Transputer Applications.* IOS Press (Amsterdam).

H. Mühlenbein and D. Schlierkamp-Voosen, 1993. Predictive models for the breeder genetic algorithm: I. continuous parameter optimisation. *Evolutionary Computing*, 1(1):25–50.

H. Mühlenbein, 1989. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann (San Mateo).

H. Mühlenbein, 1992. How genetic algorithms really work. part I: Mutation and hillclimbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2.* Elsevier Science Publishers/North Holland (Amsterdam).

A. Nix and M. D. Vose, 1991. Modeling genetic algorithms with markov chains. *Annals of Mathematics and Artificial Intelligence*, 5:79–88.

M. Norman, 1988. A genetic approach to topology optimisation for multiprocessor architectures. Technical report, University of Edinburgh.

I. M. Oliver, D. J. Smith, and J. R. C. Holland, 1987. A study of permutation crossover operators on the travelling salesman problem. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

J. Pearl, 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA.

C. C. Peck and A. P. Dhawan, 1995. Genetic algorithms as global random search methods: An alternative perspective. *Evolutionary Computation*, 3(1):39–80.

R. Poli and W. R. Langdon, 1998. Schema theory for genetic programming with one-point crossover and point mutation. (unpublished manuscript).

D. Poole, A. Mackworth, and R. Goebel, 1998. *Computational Intelligence: a logical approach.* Oxford University Press.

D. J. Powell, S. S. Tong, and M. M. Skolnick, 1989. EnGENEous domain independent, machine learning for design optimization. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

A. Prügel-Bennett and J. L. Shapiro, 1995. The dynamics of a genetic algorithm for simple random ising systems. (unpublished manuscript??).

N. J. Radcliffe and P. D. Surry, 1994a. Data mining with hierarchical genetic algorithms. Technical Report EPCC–TR94–08, Edinburgh Parallel Computing Centre.

N. J. Radcliffe and P. D. Surry, 1994b. Fitness variance of formae and performance prediction. In L. D. Whitley and M. D. Vose, editors, *Foundations of Genetic Algorithms III*, pages 51–72. Morgan Kaufmann (San Mateo, CA).

N. J. Radcliffe and P. D. Surry, 1994c. Formal memetic algorithms. In T. C. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, pages 1–16. Springer-Verlag, Lecture Notes in Computer Science 865.

N. J. Radcliffe and P. D. Surry, 1994d. The Reproductive Plan Language RPL2: Motivation, architecture and applications. In J. Stender, E. Hillebrand, and J. Kingdon, editors, *Genetic Algorithms in Optimisation, Simulation and Modelling*, pages 65–94. IOS Press (Amsterdam).

N. J. Radcliffe and P. D. Surry, 1995. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments, Lecture Notes in Computer Science, Volume 1000*, pages 275–291. Springer-Verlag (New York).

N. J. Radcliffe, 1991a. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5(2):183–205.

N. J. Radcliffe, 1991b. Forma analysis and random respectful recombination. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 222–229. Morgan Kaufmann (San Mateo).

N. J. Radcliffe, 1992a. Genetic set recombination. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann (San Mateo, CA).

N. J. Radcliffe, 1992b. Non-linear genetic representations. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 259–268. Elsevier Science Publishers/North Holland (Amsterdam).

N. J. Radcliffe, 1993. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing and Applications*, 1(1):67–90.

N. J. Radcliffe, 1994. The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence*, 10:339–384.

C. L. Ramsey and J. J. Grefenstette, 1993. Case-based initialization of genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

L. M. Rattray, 1995. The dynamics of a genetic algorithm under stabilizing selection. Technical report, Computer Science Department, University of Manchester.

T. S. Ray, 1994. An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life*, 1(1/2):195–226.

I. Rechenberg, 1973. *Evolutionstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog (Stuttgart).

I. Rechenberg, 1984. The evolution strategy. a mathematical model of darwinian evolution. In E. Frehland, editor, *Synergetics—from Microscopic to Macroscopic Order*, pages 122–132. Springer-Verlag (New York).

G. Reinelt, 1990. TSPLIB. Available by anonymous FTP from softlib.rice.edu.

J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard, 1989. Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–195. Morgan Kaufmann (San Mateo).

C. V. Russo, 1991. A general framework for implementing genetic algorithms. Technical Report EPCC–SS91–17, Edinburgh Parallel Computing Centre, University of Edinburgh.

J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, 1989. A study of the control parameters affecting online performance of genetic algorithms for function optimisation. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo).

J. D. Schaffer, 1985. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum.

J. D. Schaffer, editor, 1989. *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann (San Mateo, CA).

J. Schmidhuber, 1995. On learning how to learn learning strategies. Technical Report FKI-198-94 (revised), Technische Universität München.

M. Schoenauer and Z. Michalewicz, 1996. Evolutionary computation at the edge of feasibility. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *Parallel Problem Solving from Nature IV*, pages 245–254. (Springer-Verlag, LNCS 1141).

M. Schoenauer and S. Xanthakis, 1993. Constrained GA optimisation. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 573–580. Morgan Kaufmann (San Mateo, CA).

N. Schraudolph and R. Belew, 1990. Dynamic parameter encoding for genetic algorithms. Technical Report CS90–175, University of California, San Diego.

H.-P. Schwefel, 1981. *Numerical Optimisation of Computer Models.* John Wiley & Sons (Chichester).

C. G. Shaefer and S. J. Smith, 1990. The ARGOT strategy II: Combinatorial optimizations. Technical Report RL90-1, Thinking Machines Inc.

C. G. Shaefer, 1987. The ARGOT strategy: Adaptive Representation Genetic Optimizer technique. In *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates (Hillsdale).

J. Shapiro, A. Prügel-Bennett, and M. Rattray, 1994. A statistical mechanical formulation of the dynamics of genetic algorithms. In T. C. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, pages 17–27. Springer-Verlag, Lecture Notes in Computer Science 865.

N. Sourlas, 1986. Statistical mechanics and the travelling salesman problem. *Europhysics Letters*, 2(12):919–923.

W. M. Spears and K. A. De Jong, 1991. On the virtues of parameterised uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236. Morgan Kaufmann (San Mateo).

P. D. Surry and N. J. Radcliffe, 1994a. *The Reproductive Plan Language RPL2.* Edinburgh Parallel Computing Centre.

P. D. Surry and N. J. Radcliffe, 1994b. RPL2: A language and parallel framework for evolutionary computing. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature III*, pages 628–637. Springer-Verlag, Lecture Notes in Computer Science 866.

P. D. Surry and N. J. Radcliffe, 1996a. Formal algorithms + formal representations = search strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H. Schwefel, editors, *Parallel Problem Solving from Nature IV*, pages 366–375. (Springer-Verlag, LNCS 1141).

P. D. Surry and N. J. Radcliffe, 1996b. Inoculation to initialise evolutionary search. In *Evolutionary Computing: AISB Workshop*, pages 269–285. Springer-Verlag, Lecture Notes in Computer Science 1143.

P. D. Surry and N. J. Radcliffe, 1996c. Real representations. In R. K. Belew and M. D. Vose, editors, *Foundations of Genetic Algorithms IV*. Morgan Kaufmann (San Mateo, CA).

P. D. Surry and N. J. Radcliffe, 1997. The COMOGA method: Constrained optimisation by multiobjective genetic algorithms. *Control and Cybernetics*, 26(3).

P. D. Surry, N. J. Radcliffe, and I. D. Boyd, 1995. A multi-objective approach to constrained optimisation of gas supply networks: The COMOGA method. In T. C. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, pages 166–180. Springer-Verlag, Lecture Notes in Computer Science 993.

G. Syswerda, 1989. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

C. L. Valenzuela and A. J. Jones, 1994. Evolutionary divide and conquer (I): A novel genetic approach to the TSP. *Evolutionary Computation*, 1(4):313–334.

P. J. M. van Laarhoven and E. H. L. Aarts, 1989. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company.

M. G. A. Verhoeven, E. H. L. Aarts, E. van de Sluis, and R. J. M. Vaessens, 1992. Parallel local search and the travelling salesman problem. In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature 2*, pages 543–552, Amsterdam. Elsevier Science Publishers/North Holland.

M. D. Vose and G. E. Liepins, 1991a. Punctuated equilibria in genetic search. *Complex Systems*, 5:31–44.

M. D. Vose and G. E. Liepins, 1991b. Schema disruption. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 237–243. Morgan Kaufmann (San Mateo).

M. D. Vose, 1992. Modelling simple genetic algorithms. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann (San Mateo, CA).

S. Watanabe, 1969. *Knowing and Guessing – A formal and quantitative study*. John Wiley and Sons, Inc (New York).

D. Whitley and T. Hanson, 1989. Optimizing neural networks using faster, more accurate genetic search. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 391–396. Morgan Kaufmann (San Mateo).

D. Whitley, T. Starkweather, and D. Fuquay, 1989. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

D. Whitley, K. Mathias, and P. Fitzhorn, 1991a. Delta coding: An iterative search strategy for genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

D. Whitley, T. Starkweather, and D. Shaner, 1991b. The traveling salesmen and sequence sheduling: Quality solutions using genetic edge recombination. In L. Davis, editor, *Handbook of Genetic Algorithms*. Van Nostrand Reinhold (New York).

D. Whitley, V. S. Gordon, and K. Mathias, 1994. Lamarckian evolution, the baldwin effect and function optimization. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature III*, pages 6–15.

D. Whitley, 1989. The GENITOR algorithm and selection pressure: Why rank-based allocation of reprodutive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann (San Mateo).

L. D. Whitley, 1991. Fundamental principles of deception. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 221–241. Morgan Kaufmann (San Mateo).

D. Whitley, 1992. An executable model of a simple genetic algorithm. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann (San Mateo, CA).

N. Wirth, 1976. *Algorithms + Data Structures = Programs*. Prentice-Hall (Englewood Cliffs, NJ).

M. Wodrich and G. Bilchev, 1997. Cooperative distributed search: the ants' way. (this volume).

D. H. Wolpert and W. G. Macready, 1995. No free lunch theorems for search. Technical Report SFI–TR–95–02–010, Santa Fe Institute.

D. H. Wolpert and W. G. Macready, 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1).

A. H. Wright, 1990. Genetic algorithms for real parameter optimisation. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–220. Morgan Kaufmann (San Mateo, CA).

A. A. Zhigljavsky, 1991. *Theory of Global Random Search*. Kluwer Academic Publishers.